AD-A102 776 NATIONAL BUREAU OF STANDARDS WASHINGTON D C INST FOR--ETC F/G 9/2
XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER.(U)
JUN 81 S R KIMBLETON, P S WANG, E N FONG F30602-77-C-0066

UNCLASSIFIED RADC-TR-81-123 NL

1 OF 5
AD
A102776

LEVEL II

12

RADC-TR-81-123
Final Technical Report
June 1981

XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER

National Bureau of Standards

Stephen R. Kimbleton
Pearl S-C. Wang
Elizabeth N. Fong
Helen M. Wood
Leslie J. Miller

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

RADC-TR-81-123 has been reviewed and is approved for publication.

APPROVED: Emilie B. Jones

EMILIE B. JONES
Project Engineer

APPROVED: [illegible]

JOHN J. MARCINIAK, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-123	2. GOVT ACCESSION NO. AD-A102	3. RECIPIENT'S CATALOG NUMBER 776
4. TITLE (and Subtitle) XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report February 77 — September 80
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Stephen R./Kimbleton, Helen M./Wood, Pearl S-C./Wang, Leslie J./Miller, Elizabeth N./Fong		8. CONTRACT OR GRANT NUMBER(s) F30602-77-C-0066
9. PERFORMING ORGANIZATION NAME AND ADDRESS National Bureau of Standards Institute for Computer Sciences and Technology Wash DC 20234		10. PROGRAM ELEMENT PROJECT TASK AREA & WORK UNIT NUMBERS 62702F 55812127
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCP) Griffiss AFB NY 13441		12. REPORT DATE June 1981
		13. NUMBER OF PAGES 408
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Emilie B. Jones (ISCP)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

ARPANET	Distributed Databases
UNIX	Uniform User Interface
Network Operating Systems	Common Query
Resource Sharing	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report documents the three year investigation by the National Bureau of Standards into the technical issues involved in providing a uniform user and program environment for (possibly concurrent) access to multiple heterogeneous remote database management systems. This report is an anthology of papers prepared by the investigators which identify and discuss the design, development and implementation of such a user interface, and the development status of a demonstrable prototype.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EVALUATION

The Experimental Network Data Manager Final Report provides a technical perspective for developments in the uniform user data manipulation interface area as it applies to distributed heterogeneous computer networks. The format is that of an anthology of technical papers prepared by the investigators. This collection of papers identifies and discusses the technical issues in the design, development, and implementation of such an interface system and the development status of a demonstrable prototype.

This effort directly supports the Database Subthrust (4G2) of the Information Processing Technical Planning Objective. The results from this effort have been and will be used as background and guidance in formulating the continued technical program in Information Processing Technology.

Emilie B. Jones

EMILIE B. JONES
Project Engineer

I

OVERVIEW

Excerpted from NBS FY80 Proposal

Edited by
Emilie B. Jones
Rome Air Development Center

1.0 INTRODUCTION

Computer Scientists, in the early 70's, recognized the immense amounts of data that were necessary in order to maximize the efficiency of an organization in carrying out its assigned mission. As a result of this perception, a substantial degree of interest in Very Large Data Bases (VLDBs) has developed. Although the amounts of data requiring support were, indeed, large, this appears to be the wrong technology for many, if not most applications of interest to the organization. Major problems working against the establishment of a Very Large Data Base are: i) technology trends, ii) organizational differences, iii) resource sharing, iv) legislation and v) environmental dynamics.

Because of technology trends, specifically the declining cost of hardware, the increasing cost of constructing software, and the increasing availability of networking technology, organizational acceptance of the rigidity implicit in adhering to the requirements (data model, date structures, and data manipulation language) of a single VLDB is unlikely. Moreover, given the need, within the competitive marketplace, to use advancing technology as a way of reducing costs, it seems unlikely that practical acceptance of VLDB's will increase.

A variety of authors, including Professor J. Saltzer of MIT, have observed that the declining cost of computers renders them subject to procurement at lower levels of the organization. The natural tendency in such a procurement is to optimize the match between the procured system and the present needs of that unit of the organization. Since these needs differ, computer system architectures will also differ as will the database management systems running thereon.

Resource sharing is an essential requirement for distributed organizations since different components may occasionally need to access data maintained by other components. For example, strategic planning functions may need to browse operational data maintained by payroll, inventory, and other organizational functions.

For the federal sector, legislation exists requiring procurement from the lowest cost bidder. The result of this legislation, given the fluctuating fortunes of organizations, is the likelihood that procurements in different years will result in different systems. Thus, legislation is a strong forcing function toward heterogeneity.

Finally, although the observation that the environment is rapidly changing "out there," is less than deep, it should be noted that his observation does have a very strong implication. In particular, it implies that the effective organization will be one which can adapt and track this rapidly changing environment. Since this tracking operation will usually require information processing support, a more specific implication is the need for constructing adaptable systems in the face of the preceding five heterogeneity forcing functions.

In recognition of the needs outlined above, RADC and NBS have funded a joint project directed toward development of an Experimental Network Data Manager (XNDM). Although FY 79 was the first fiscal year for which funding was devoted toward development of XNDM, this project was logically an extension of an earlier joint project concerned with development of a Experimental Network Operating System (XNOS). XNOS provides essential support components for the operation of XNDM through facilitating inter-system communication.

The primary objective of XNDM is providing a uniform user and program environment for (possibly concurrent) access to multiple heterogeneous remote database management systems. Figure 1 logically illustrates the XNDM objective.

Based upon work done during FY-79 and FY-80 it has been demonstrated to a variety of audiences, that providing this uniform user viewpoint for queries is technically feasible. However, a substantial amount of work remains to be done,including the extension of present partial results to cover the six major query language categories, for Codasyl DBTG DBMs, and the complete redesign of the current partial translator based upon insights developed through implementation of the first version.

In order to provide an appropriate perspective for these objectives, a certain amount of background information is required. This information will now be provided.

2.0 EXPERIMENTAL NETWORK DATA MANAGER (XNDM)

The global framework supporting the uniform user viewpoint across multiple, remote, network accessible DSMSs, is termed an Experimental Network Data Manager (XNDM). The basic structure of XNDM has been described in detail (KIMBS 79).

Structurally, XNDM consists of three major components i) the network user's view of data, ii) translators between this view and the target DBMSs, and iii) encapsulation of the target DBMs to accommodate structural differences within a given date model class and to minimize network traffic. The network user's view is currently well defined and will be summarized below. Translation technology was the major issue being addressed in the RADC funded tasks. The precise structure of the

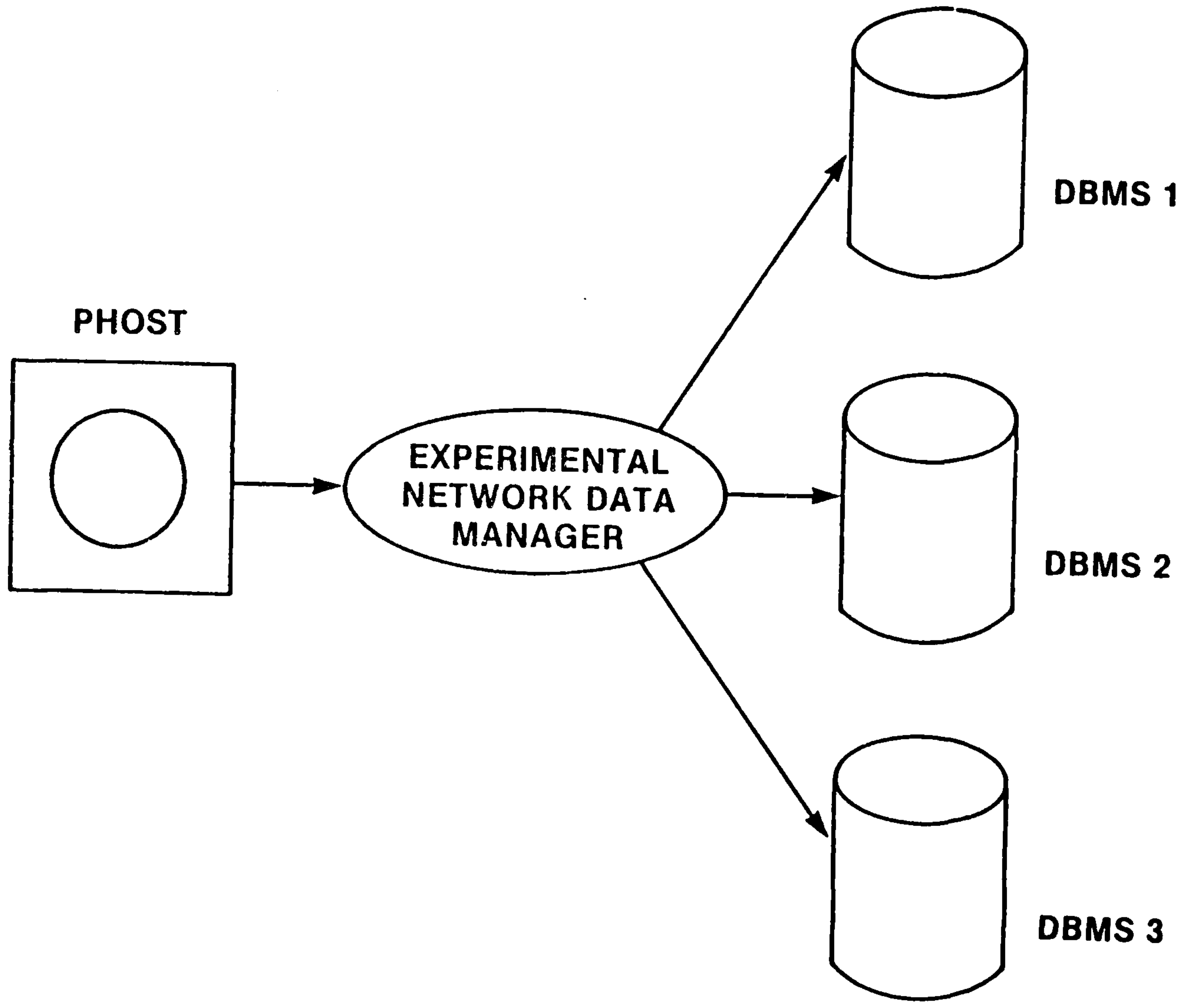


FIGURE 1. LOGICAL VIEW OF AN NVDM

local DBMS encapsulation is still being investigated. However, substantial basic progress has been made and an initial encapsulation for Codasyl-DBTG DBMS has been implemented. The issue of performance has not yet been investigated in any depth. It should be noted that, given current knowledge it is clear that a limited number of "hooks" in the target DBMS are desirable to simplify the interfacing problem. (For example, the cited DBTG DBMS requires that the user specify field widths exactly. This complicates the interfacing mechanism since it does not permit utilization of a default value.)

2.1 The Network User's Viewpoint

Three key XNDM design decisions determining the network user's view of data are: flexibility, selection of a data model, and specification of a data language.

2.1.1 Flexibility

Perhaps the most important design decision was the goal of flexibility. This goal reflects our perception that the current state-of-the-art in database technology is incapable of operationally defining an "optimal" network user's view of data. Only a limited amount of database related human factors work has been performed, new data models are being developed, semantic integrity is still in an explorational stage, and only a limited amount of experience with access control systems has been derived. As a result, two principles have guided the original design: i) conceptual simplicity and ii) extensibility. Moreover, implementation has been guided by the principle of flexibility. The major result of this principle is the design of the XNDM implementation as a discrete set of modules interacting in very carefully defined ways. As a consequence, changes in a given XNDM functional component usually only require changes in the corresponding module(s).

The principle of conceptual simplicity reflects the assumption that the network user does not have prior knowledge of the characteristics of the systems being accessed. Moreover, since the requirement for accessing remote systems is assumed to be unpredictable and non-repetitive, minimizing access difficulties, possibly at the cost of a performance penalty, seems reasonable.

The principle of extensibility reflects the fact that there are few hard truths in contemporary database research. As a result, providing a flexible, extensible, and easily adaptable approach seems preferable to attempting to single out a "best" approach.

2.1.2 XNDM Data Model

The XNDM Data Model decision was driven by the desire to focus on system rather than component issues. Consequently, one of the major implemented data models was chosen in preference to some of the newer models in the recent literature or to developing another one. This decision facilitates describing XNDM to potential users. Additionally, frontends supporting other data models can be interfaced to XNDM as the desirability of doing so becomes evident.

Currently, three primary data models are used to define global schemas: relational, hierarchial and CODASYL. In selecting among these three data models, the major decision was that the complexity of the data model should be evaluated in terms of the perceived complexity of the resulting global schema rather than of a user schema or view. This reflects the observation that while repetitive or high bandwidth applications usually merit a custom-tailored user schema, the unpredictable, non-recurrent requests which XNDM was intended to support preclude the development of user-schemas. As a result, the user or application would be required to interact with a global schema or a substantial component of such a schema, e.g. a payroll component.

Consequently, tables (relations) were chosen as the basic data model for XNDM. This basic model is to be enhanced with access controls and semantic integrity capabilities.

Since custom-tailored user views are precluded, the relational data model provides a conceptually simpler data model than the hierarchial or CODASYL data models. Moreover, since the data is actually resident on target computer systems which are not under the control of the XNDM managers (DBA's), storage efficiency is not a concern.

2.1.3 XNDM Data Language

Having selected the relational data model, the appropriate data language is one providing the power of the relational calculus. SQL (SEQUEL) was chosen as the basic framework for the Experimental Network Data Language (XNDL). This decision was driven by three factors. The first was the previously cited objective emphasizing systems rather than component issues. The second was the completeness of the specification of SQL in terms of a data definition language, data manipulation language, and data control language. The third was based on the operational fact that System R is not currently Arpanet accessible. As a result, since the Arpanet is being used to support communication, selecting SQL would not restrict the number of systems which could potentially be supported by XNDM.

Specification of the DDL and DCL portions of XNDL has been deferred pending completion of the specification of the DML. Currently, this DML provides primitives in six major categories as illustrated in Table 1. These six categories comprise a subset of those provided by SQL. Specifically, a host language interface is not provided, certain redundant means for specifying predicates have been eliminated, and sorting is not supported.

Since XNDM supports concurrent access to multiple remote DBMSs an extension to the SQL DML was required to support the specification of the target system(s). Three types of specifications have been structured: i) explicit specification by the user, ii) implicit specification of all systems containing the specified data element(s), and iii) specification through using location as a virtual attribute. (That is, a location attribute is virtually attached to each relation corresponding to a given target system.) This third capability is designed in supporting relatively complex expressions describing when data is to be accessed. For example, an out-of-stock replenishment rule may be dependent upon both distance (to minimize shipping costs) and stock on hand at the sites from which replenishment is being performed.

Selection of a relational calculus DML is a reasonable step toward achieving the goal of minimizing procedurality. Of course, any relationships not explicitly specified within the schema must be explicitly specified by the user. As a result, the goal of being non-navigational is at best only partly achieved.

Currently, a significant amount of research in minimizing the navigational requirement in interacting with table based schemas is being conducted. Chen (CHENP 76) has developed the entity-relationship model which, fundamentally, is a graph describing how different table-based data structures are logically interrelated. McLeod (MCLED 79) has developed a semantic data model which structures the interrelationship between attributes in a way which is intended to be helpful to the network user. Since XNDM is being implemented in a flexible way, it should prove feasible to provide navigational support at a later time. Currently, such support is not provided.

2.2 Translation Technology

Given that the netowrk user's view is based upon the relational data model, the requirement exists for translating between this view and the data structures actually maintained by the target DBMSs being accessed. This translation process is a major technical problem. As a result, developing a basic structure for performing such translations was a major technical concern during FY 79. Since this issue has been addressed at length in a published paper (KIMBS 79) it will not be discussed further.

SELECT

SELECT. . . WHERE

AGGREGATION

PARTITION

SET OPERATIONS

COMPOSITION

Table 1. DML PRIMITIVE CATEGORIES

2.3 Local DBMS Encapsulation

During the construction of the first version of XNDM, it became apparent that close attention should be paid to the design of a proper interface between XNDM and LDBMSs. By encapsulating LDBMSs within a software module which implements an appropriately designed set of functions, the very desirable goal of uniformizing the XNDM-LDBMS interfaces can be achieved. This reduces, by a very significant amount, the high cost of installing new DBMSs into the network, it also confines the effects caused by modifications and enhancements to the individual local DBMSs and thereby reduces the high cost of maintenance for the system.

Currently, only very crude ideas about the design and the functionality of the interfaces have been formulated.

3.0 PROTOCOL WORK

Performance of the RADC supported activities was complemented by the execution of NBS supported tasks in two major areas: protocol specification and verification and protocol performance modeling.

3.1 Protocol Specification and Verification

Protocol specification and verification is concerned with providing a precise and unambiguous means for stating the functions to be provided for a protocol toget her with a mechanism for verifying that the stated functions are, indeed, correctly provided.

Current work in protocol specification and verification is primarily concerned with lower level protocols such as X.25 and host-host protocols such as TCP/4. Such protocols involve asynchronous, distributed, and concurrent processes. Their correct specification and verification is proving fundamentally hard.

Many higher level protocols such as Data Transfer Protocols, Distributed Job Execution Protocols, and Network Data Access Protocols have a transaction oriented flavor. Asynchrony and concurrency requirements are substantially reduced. It is reasonable to ask if this can result in substantial simplification of the difficulties in their specification and verification. NBS personnel believe this to be the case. Work currently being documented at NBS provides the appropriate supporting evidence for a particular Structured Data Transfer Protocol.

3.2 Protocol Performance Modeling

Currently, a variety of higher level protocols have been identified. Morever, it has also become accepted to view protocols in terms of a layered implementation. Thus, these higher level protocols (file transfer, remote job entry, network interprocess communication, network distributed data support, common command languages and others) are implemented assuming a host-host protocol which, in turn, assumes appropriate lower level protocols such as X.25. X.25 in turn, can be implemented in either a virtual circuit of datagram mode.

It is clear that there will be substantial interaction among these layers. It is not at all clear what the performance impact of this interaction will be. However, by now, it is also clear that it is necessary to have some understanding of these performance implications since, for protocols, poor design is usually reflected in poor performance, e.g. one or two orders of magnitude less than expected, rather than complete failure.

With this perspective in mind, a second major dimension of the NBS supported activities was directed toward undertaking a performance investigation of protocols stressing the interactions among them rather than the detailed structure of any individual protocol. This investigation is crucial to the successful adoption of a layered approach as a means of supporting substituability and extensibility. Accordingly, an initial scoping of the dimensions of this problem was undertaken by Dr. Leslie Jill Miller. (MILLL80).

4.0 CONCLUDING REMARKS

With the departure of the two principal investigators during the third quarter of FY-80, the work at NBS was terminated.

This report is in the form of an anthology of technical papers prepared by the investigators during the last year of the effort. While there is some duplication of information, each paper, nevertheless, contributes toward understanding this particular approach to the problem.

5.0 REFERENCES

(CHENP76) Chen, P. P-S., The Entity-Relationship Model--Toward a Unified View of Data, ACM Transactions on Database Systems, Vol. 1 (1976).

(KIMBS79) Kimbleton, S. R., et al., XNDM: An Experimental Network Data Manager, Proc. Fourth Berkeley Conference on Distributed Data Management and Computer Networks, August 1979.

(MCLED79) McLeod, D. and King, R., Applying a Semantic Database Model, University of Southern California, Computer Science Dept. TR No. 5, Los Angeles CA, August 1979.

(MILLL80) Miller, L. J. and Kimbleton, S. R., The Effects of Communications Subnet Choice Upon Host-Host Protocol Performance, NBS, 1980.

II

DATA ACCESS PROTOCOLS: OBJECTIVES, PRINCIPLES, AND AN IMPLEMENTATION APPROACH

Stephen R. Kimbleton

Pearl S-C. Wang

National Bureau of Standards

March 1980

ABSTRACT

Network database access promises to be a major application of computer communication networks. Accessing multiple, heterogeneous, remote Database Management Systems promises to be difficult because of differences in: i) the data model employed, ii) data structures constructed using this data model, iii) Database Management System functional differences, iv) differences in Data Manipulation Languages used in interacting with the data structures, and v) computer system differences. Imposing a uniform user viewpoint across this collection of differences is an alternative to requiring detailed user knowledge of the characteristics of each of the accessed systems. Network Virtual Data Managers are intended to provide this capability. This paper structures the general issues in their implementation including: a) the virtual environment presented to the network accessor, b) the translation technology required to support this virtual environment, and c) the methodology required for preservation of meaning in returning structured data generated by individual local Database Management Systems to the network user. This discussion is supported by a description of relevant prototype components implemented at the National Bureau of Standards.

1. INTRODUCTION

Computer communication networks permit users to access remote programs and data. Such access generally proves difficult because of differences in target systems and Database Management Systems (DBMSs) residing on them. Conversion costs usually preclude the obvious solution of forcing all systems and application packages such as DBMSs to conform to a common set of access and use conventions.

Protocols have been developed to support basic communication between both users and network accessible systems as well as between different systems. Extensive exploratory work has established the functional objectives of a collection of basic protocols [ISO 79]. Currently, a major (U.S.) Government investment is being made to develop the corresponding protocol standards.

Currently existing protocols do not provide a unified framework for accessing and retrieving remote data. Since an organization's data is increasingly recognized as one of its most valuable resources, the need for such a framework is evident. This section structures an initial approach for its realization.

Structuring data access protocols requires the background information on DBMSs provided in section 2. Section 3 establishes basic issues in providing a uniform user viewpoint across multiple remote heterogeneous DBMSs.

Implementing such a viewpoint is discussed at length in section 4.

Given that a query or update has been processed by the target DBMSs, the resulting individual outputs must be aggregated and the collective result returned to the requesting process. This requires a protocol for preserving the meaning of data being transmitted between heterogeneous systems; the nature of the required data transfer protocol is discussed in section 5.

Sections 2-5 scope the problem of supporting remote access to data. The merits of this approach depend on its ultimate implementability. Currently, prototype components have been implemented at the National Bureau of Standards. Accordingly, sections 3.4 and 5.5 contain concise overviews of an Experimental Network Data Manager (XNDM) and an Experimental Network Operating System (XNOS). The XNDM provides uniform user access across remote heterogeneous DBMSs. The XNOS supports XNDM by providing a uniform mode of user and system interaction across systems.

1.1 Supporting Program Access to Data

Traditional approaches to supporting program execution require co-location of the program and its data. In a networking environment, this was usually accomplished by bringing the program to the data or the data to the program.

Two factors mitigate against the general applicability of this approach. The first is the difficulty of producing truly portable programs in view of the large amount of information required for characterizing the program execution environment, coupled with the tendency of vendors to produce similar, but nevertheless incompatible compilers. The second is the increasing use of Database Management Systems, a notably nonportable source of data.

Organizational strategic planning and exception reporting are important information processing functions whose support, in a networking environment, requires the ability to request data from remote databases and a mechanism for preserving the structure of the response being transmitted between systems. This is a problem of manifest importance; a general solution is therefore of interest.

Current teleprocessing techniques permit users to retrieve data from remote databases. A straightforward extension would support retrieval of data from multiple remote heterogeneous DBMSs. The implicit requirement is user knowledge of the properties of each of the accessed databases. Given the complexity of an individual DBMS, the resultant learning burden is likely to be sufficient to preclude effective routine access to multiple remote DBMSs. A different approach is required. We believe that it should mask DBMS differences from the user. Providing such a mask is a major problem, since the spectrum of differences includes: i) the underlying data model, ii) the data structures developed using this data model, iii) Data Manipulation Languages used for manipulating these data structures, iv) functionality provided by the DBMSs, even if the underlying data model is the same, and v) the computer systems on which the DBMSs reside.

1.2 Distributed Applications

Currently existing protocols primarily support two party interactions in which one party is a system and the other party is a user or system. Distributed applications, in contrast, require n party interactions.

The difficulty of constructing a distributed application depends upon whether individual components existed prior to initiating application construction. If so, differing communications conventions are likely and translation technology is required. Otherwise, these differences can be eliminated through requiring individual components to conform to conventions established at application design time.

Although forced uniformity eliminates the need for translation technology, we doubt its merits. This reflects the perception that future needs can rarely be anticipated in a dynamic area like computer communications. The continuing stream of new products and capabilities is a strong forcing function for heterogeneity; hence an appropriate translation technology will always be required. The following demonstrates the feasibility of using translation techniques to support a large class of remote database query applications.

2. DATABASE MANAGEMENT SYSTEMS (DBMSs)

Declining computing costs permit more effective information processing support for users and organizations through requiring less efficient use of these resources. The increasing interest in and sophistication of DBMSs reflect this. This section provides a brief overview stressing needs, categories, data models, and data manipulation languages.

2.1 The Need for DBMSs

DBMSs effectively separate the logical and storage structures of data. Thus, users can access data by name and logical interrelationships rather than by location. As a result, the way in which data is physically stored can be modified without affecting program access. Moreover, the same data can be shared by several programs. (Additionally, a DBMS may provide certain guarantees, e.g., access controls and security, as well as useful utilities such as report writers. The Codasyl Systems Committee [Codasyl 76], particularly chapter 3, contains a useful overview of DBMSs.)

Data sharing is facilitated through using schemas. The ANSI/SPARC committee on DBMSs [ANSI/X3/SPARC 75] has identified three major schema levels: external, conceptual, and storage (internal). (Not all existing DBMSs possess exactly these three schemas.) External schemas describe the user or user program view of data. Conceptual schemas define the corporate or organization-wide view of data. Internal schemas establish access paths to the data.

These three schemas can be viewed as layers through which a request passes in actually accessing data managed by a DBMS. Consequently, processing a request requires mapping the external schema to the conceptual schema which, in turn, is mapped to the internal schema.

DBMS design poses three major trade-offs:

- performance vs. understandability
- performance vs. adaptability
- user effort vs. system effort

DBMSs differ in their ability to support easily understood logical, i.e., user perceived, data structures. Ease of understanding usually results in reduced efficiency in accessing and storing data.

Logical data structures can be tailored to facilitate processing if the nature of the request is known. This yields better performance. It may also result in poorer performance when processing unpredictable queries involving the same data.

DBMS access can be supported at two polar extremes. The simplest, for the DBMS implementer, views the DBMS as the repository of data and requires explicit user issuance of all retrieval commands. The simplest for the user requires only specification of what data is to be retrieved and not how. Ease of use is gained at the expense of both increased processing and the requirement for a more sophisticated system. Declining systems costs facilitate simpler user interfaces. In any case, the processing of repetitive requests should probably be optimized.

2.2 DBMS Differences

DBMSs can be categorized in terms of the data model which is supported and the data language provided for interacting with this data model. A data model defines the acceptable types of data structures. The three basic data models discussed below are logically based on the representation of data as record types; the network and hierarchical models also support explicit system maintenance of record type interrelationships.

Data languages can be divided into three major components: Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL). DDLs are used to define the data structures, their components and interrelationships. DCLs permit specification of whatever access controls are provided by the DBMS. DMLs provide the means for issuing queries or updates against the database. Thus, the DDL and DCL define the organization's view of data; the DML permits user interaction with data. Discussing DML alternatives requires a fuller understanding of the alternative data models.

2.3 Data Models

A record of type R is defined to be a linear sequence of fields, that is:

R: f1:T1; . . .; fn:Tn.

Two records are of the same type if the sequences of component field types Ti are identical. A record type defines the formal structure of a record. Both individual instances of a record type as well as the cardinality of the collection of instances of a record type will usually vary with time.

The three basic data models assume an underlying collection of record types. They differ in their maintenance of record type interrelationships.

2.3.1 The Relational Data Model

The relational data model [Codd 70] represents data as tables. (In this section, we shall carefully distinguish between the representation of data and any semantics attached to that representation.) Logically, a relation is a collection of instances ("tuples") of a record type in which the sequencing of the instances and the sequencing of the fields within the record type are unimportant. Thus, two relations are equivalent (i.e., they have the same "intention") if the sequence of field names and corresponding types of one relation is a permutation of the field names and types of the other, and they are identical (i.e., they have the same "extension") if the sets of tuples they contain are identical.

Representing data as tables simplifies understanding the structure of the database. Logically, these tables describe entities and interrelationships among them. Interrelationships which are not represented as tables can be explicitly specified by the user via JOINs. Using JOINs provides a very flexible means for specifying interrelationships. The disadvantage is the possibility of stating semantically meaningless interrelationships, e.g., a JOIN on the domain SNAME in the Supplier relation and the domain PNAME in the Parts relation of figure 1. Moreover, processing queries or updates involving user specified, semantically meaningful interrelationships may yield unsatisfactory performance if the tables are large and their expression was not anticipated.

SUPPLIER

SNO	SNAME	SSTAT	SCITY
1	Chen	Bad	Washington
2	Chu	Good	Boston
3	Lawrence	Good	Chicago
4	Jones	Check	Phoenix
5	Smith	Good	Boston

PARTS

PNO	PNAME	PCOL	PWT	PLOC
1	Screw	Black	24.0	Washington
2	Nut	Brown	30.8	Rome
3	Bolt	Blue	15.5	Boston
4	Screw	Black	30.8	Washington
5	Nut	Blue	26.4	Boston
6	Bolt	Red	10.0	Chicago

STOCK

SNO	PNO	QUANT
1	1	30
1	2	20
2	2	40
2	4	20
2	5	100
3	1	100
3	4	30
3	5	40
3	6	20
4	2	20
4	4	30
4	5	40

FIGURE 1. SUPPLIER-AND-PARTS RELATIONAL SCHEMA

Figure 1 illustrates a relation called Supplier, containing four columns: SNO (supplier number), SNAME (supplier name), SSTAT (supplier status) and SCITY (location of the supplier). This relation is our representation of the real-world entity "supplier". Another relation, Parts, contains relevant information concerning the entity "parts": PNO (part number), PNAME (part name), PCOL (part color), PWT (part weight) and PLOC (where the part is stored). Information about the interrelationship between suppliers and parts is stored in the relation Stock: it tells us who (SNO) supplies what (PNO), and also the amount of the supply (QUANT). The corresponding schema definition (in Multics Relational Data Store (MRDS) syntax [Honeywell 78]) is given in figure 2. The first part of this schema definition consists of declarations of the field ("domain") names and their data types, the second part defines the tables ("relations") in terms of their component fields.

```
domain:   SNO       fixed bin,
          SNAME     char(15),
          SSTAT     char(8),
          SCITY     char(20),
          PNO       fixed bin,
          PNAME     char(15),
          PCOL      char(12),
          PWT       float bin,
          PLOC      char(20),
          QUANT     fixed bin;

relation: Supplier  (SNO SNAME SSTAT SCITY)
          Parts     (PNO PNAME PCOL PWT PLOC),
          Stock     (SNO PNO QUANT);
```

Figure 2. Supplier-and-Parts Relational Schema Definition

2.3.2 The Hierarchical Data Model

The hierarchical data model [Date 77] permits data structures which are logically equivalent to a tree of record types. (Record types are often termed segments within the hierarchical data model.) Thus, specified hierarchical interrelationships between one or more record types are explicitly maintained by the system. Actually, the interrelationship is really between a record instance at a given level in the hierarchy and a collection of corresponding record instances at the next lower level in the hierarchy.

Because of the additional interrelationship structure maintained by the hierarchical data model, processing of predictable queries is facilitated. Processing of queries involving interrelationships not explicitly defined within this data model prove correspondingly more difficult.

As an example, we again consider the Supplier-and-Parts database discussed before. One possible hierarchical structure of this database is illustrated in figure 3. Here we have four types of records (segments): Supplier, Sstatus, Stock and Parts. Supplier is the root segment-type, it is the parent of Sstatus and Stock segment-types. The latter segment-type (Stock) is in turn the parent of the Parts segment-type. Figure 4 shows a sample occurrence graph for this database. It illustrates three important features of the hierarchical data model: first, each parent record occurrence can have a varying number of child occurrences connected to it, for example, the specific Supplier occurrence being considered (SNO=4) has one child Sstatus occurrence and three child Stock occurrences. Second, there is a single type of root segment (Supplier) and it can have any number of child segment-types (Sstatus and Stock). Third, each child of the root can have any number of child types, and so on, and all record occurrences must be connected to a parent unless they belong to the root segment type. For illustrative purposes, we also show (figure 5) the schema definition for this database, using IBM's Information Management System (IMS) database description [IBM 76] as our DDL. The first statement of this definition file assigns the name Supplier-and-Parts to the DBD ("database description"), the SEGM statements define the segment names and their lengths

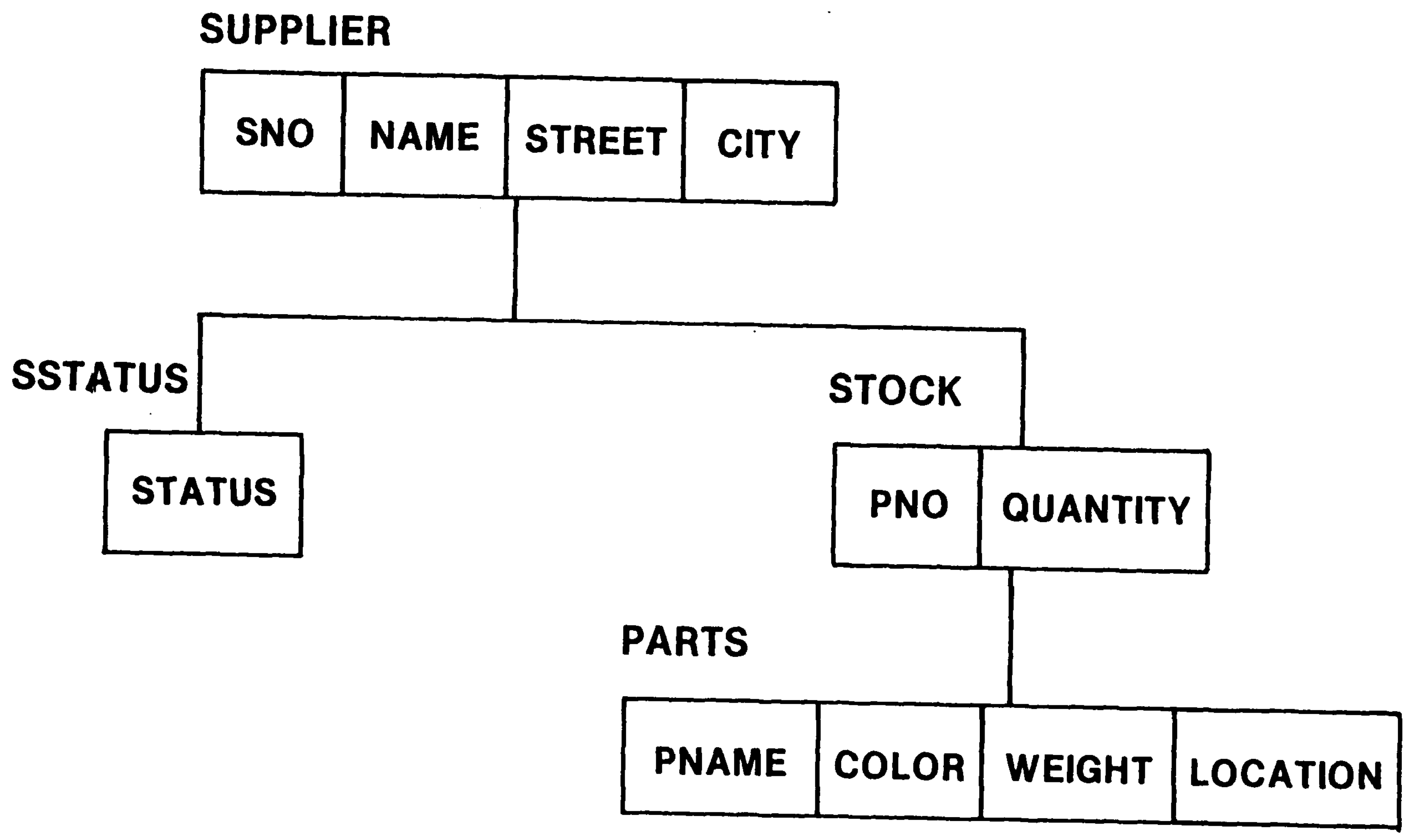


FIGURE 3. SUPPLIER-AND-PARTS HIERARCHICAL SCHEMA

(in bytes) and are followed by FIELD statements giving the constituent field names, field lengths and their starting positions within the segment. The reader is referred to the IMS documentation [IBM 76] for a complete explanation of this DDL.

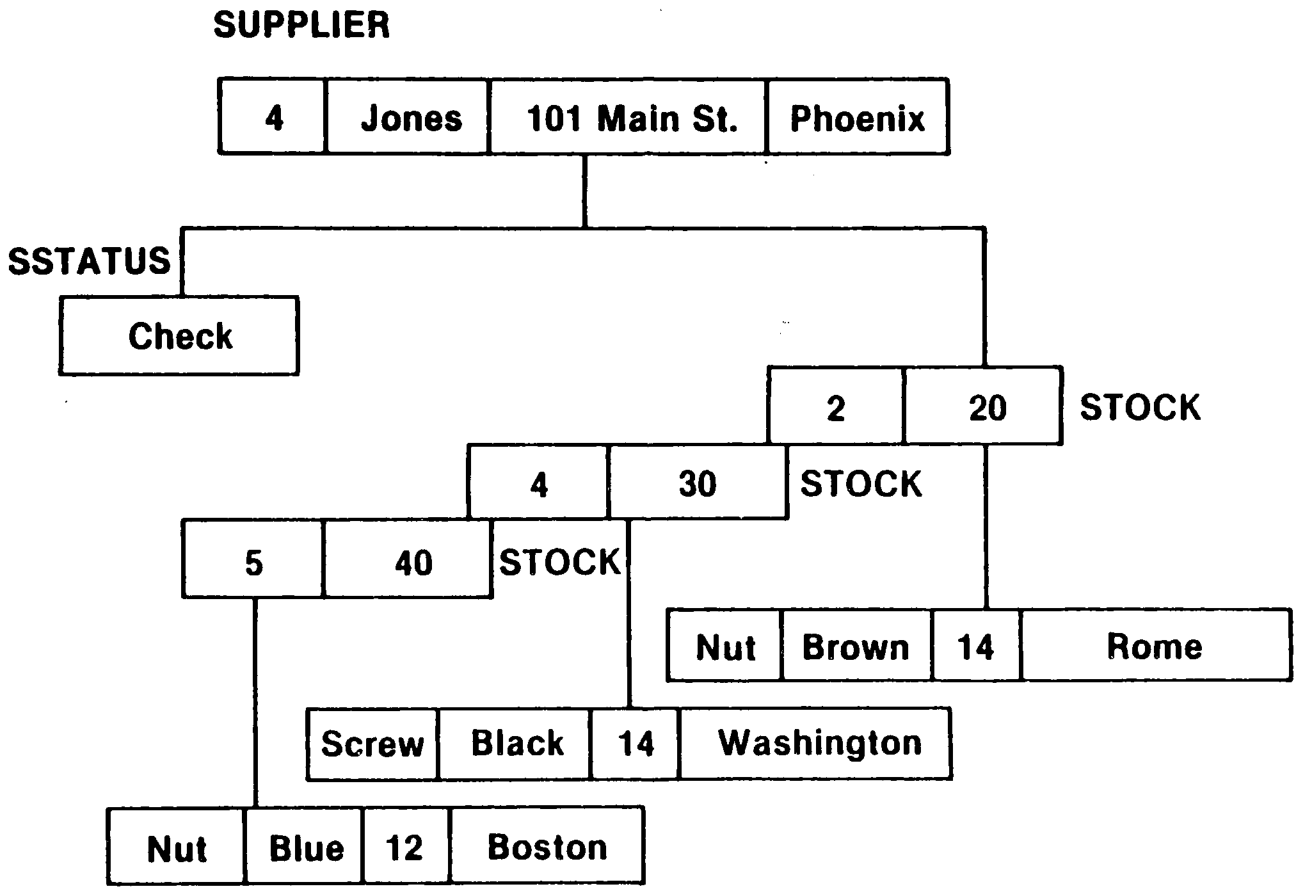


FIGURE 4. SUPPLIER-AND-PARTS HIERARCHICAL OCCURRENCE GRAPH

```
DBD      NAME=Supplier-and-Parts
SEGM     NAME=Supplier,BYTES=96
FIELD    NAME=(SNO,SEQ),BYTES=2,START=1
FIELD    NAME=NAME,BYTES=15,START=3
FIELD    NAME=STREET,BYTES=32,START=18
FIELD    NAME=CITY,BYTES=32,START=50
SEGM     NAME=Sstatus,PARENT=Supplier,BYTES=8
FIELD    NAME=STATUS,BYTES=8,START=1
SEGM     NAME=Stock,PARENT=Supplier,BYTES=8
FIELD    NAME=(PNO,SEQ),BYTES=4,START=1
FIELD    NAME=QUANTITY,BYTES=2,START=5
SEGM     NAME=Parts,PARENT=Stock,BYTES=64
FIELD    NAME=PNAME,BYTES=8,START=1
FIELD    NAME=COLOR,BYTES=12,START=9
FIELD    NAME=WEIGHT,BYTES=4,START=21
FIELD    NAME=LOCATION,BYTES=20,START=25
```

Figure 5. Supplier-and-Parts Hierarchical Schema Definition

2.3.3 The Codasyl Data Model

The Codasyl Database Task Group (Codasyl DBTG) or network Data model [Codasyl 73] (we shall avoid the use of this latter term for obvious reasons and use either Codasyl or DBTG to refer to this model hereafter) structures data as directed graphs of record types. As with the hierarchical data model, the actual (DML level) interrelationship is between record instances rather than between types. The graph is constructed using the set concept. A set consists of two record types: owner and member. Although owner and member record types cannot be the same, a given record type can occur more than once in an access path. (A more detailed discussion of this point can be found in section 2.4.)

The Codasyl data model yields conceptual schemas which are relatively complex and therefore difficult to understand. However, this complexity permits efficient use of storage space. Intuitively, it also permits efficient processing of a wider range of query or update types.

A possible DBTG schema for the Supplier-and-Parts database is shown in figure 6. The database is organized into three record types (Supplier, Parts and Stock). These are related through the sets S-S (owner Supplier, member Stock) and P-S (owner Parts, member Stock). The sets A-S (owner All-Supplier, member Supplier) and A-P (owner All-Parts, member Parts) serve the function of collecting all Supplier and Parts records for sequential access. (For systems supporting full DBTG capabilities, we could have eliminated the need for the record types All-Supplier and All-Parts by making A-S and A-P singular.) Figure 6 shows the data structure diagram [Bachman 69] for this database. Part of the actual data model is shown in figure 7. The schema itself is given in figure 8 in Multics Integrated Data Store (MIDS) [Honeywell 77] schema definition language. A brief explanation of this figure is as follows: the first statement assigns a name to the schema. The RECORD statements declare the record-type names and their various properties (for example their LOCATION MODEs) for purposes of controlling the storage and location of record occurrences. The SET statements define the set-types: their names, their owner and member record-types, the manner in which the individual set occurrences should be sequenced (the ORDER clause) and the access strategies to be associated with the set (the SET SELECTION clause). A more detailed description of this DDL can be found in Honeywell Information Systems documentation [Honeywell 77] and in Codasyl documentation [Codasyl 73].

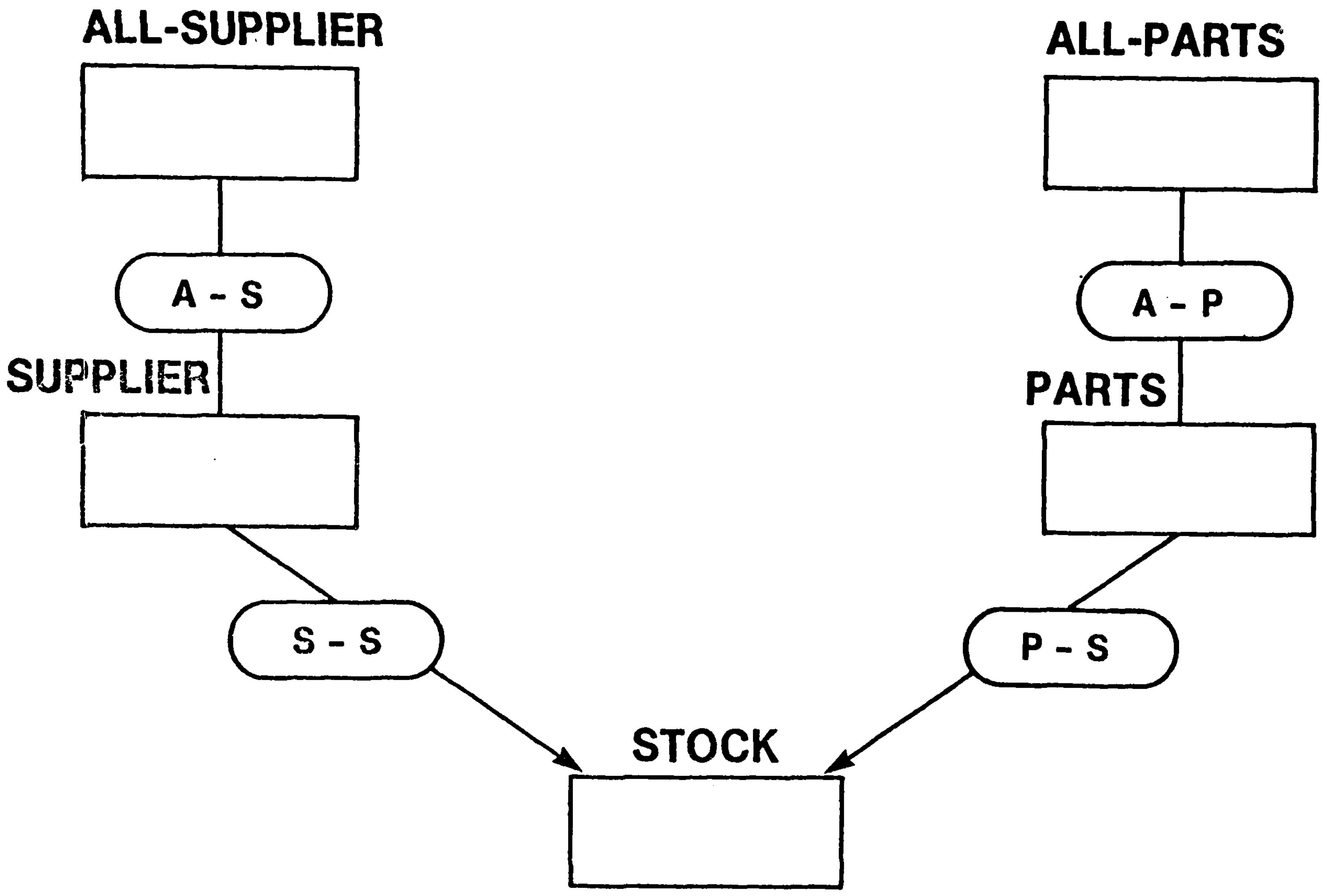


FIGURE 6. SUPPLIER-AND-PARTS CODASYL SCHEMA

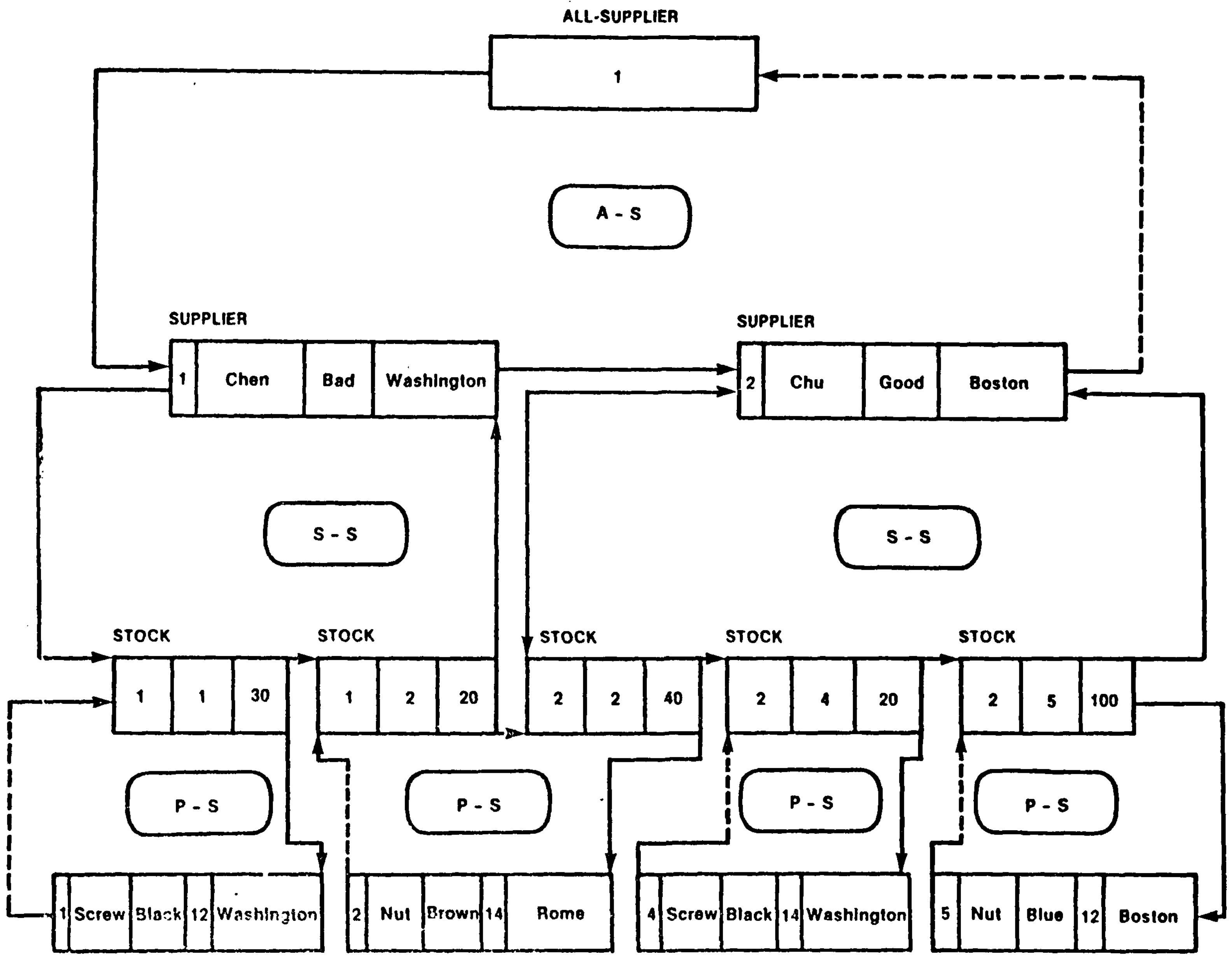


FIGURE 7. SUPPLIER-AND-PARTS CODASYL OCCURRENCE GRAPH

```
SCHEMA Supplier-AND-Parts.
RECORD All-Supplier;
  LOCATION CALC USING SKEY.
  02 SKEY; TYPE DECIMAL 5.
RECORD All-Parts;
  LOCATION CALC USING PKEY.
  02 PKEY; TYPE DECIMAL 5.
RECORD Supplier;
  LOCATION VIA A-S.
  02 SNO; TYPE DECIMAL 5.
  02 SNAME; TYPE CHARACTER 15.
  02 STATUS; TYPE CHARACTER 8.
  02 CITY; TYPE CHARACTER 20.
RECORD Parts;
  LOCATION VIA A-P.
  02 PNO; TYPE DECIMAL 5.
  02 PNAME; TYPE CHARACTER 15.
  02 COLOR; TYPE CHARACTER 12.
  02 WEIGHT; TYPE DECIMAL 5.
  02 LOCATION; TYPE CHARACTER 20.
RECORD Stock;
  LOCATION IS SYSTEM-DEFAULT.
  02 SNO; TYPE DECIMAL 5.
  02 PNO; TYPE DECIMAL 5.
  02 QUANTITY; TYPE DECIMAL 5.
SET A-S;
  OWNER All-Supplier;
          ORDER PERMANENT NEXT.
  MEMBER Supplier;
          MANDATORY AUTOMATIC LINKED TO OWNER;
  KEY ASCENDING SKEY;
SET A-P;
  OWNER All-Parts;
          ORDER PERMANENT NEXT.
  MEMBER Parts;
          MANDATORY AUTOMATIC LINKED TO OWNER;
  KEY ASCENDING PKEY;
SET S-S;
  OWNER Supplier;
          ORDER PERMANENT NEXT.
  MEMBER Stock;
          MANDATORY AUTOMATIC LINKED TO OWNER;
  KEY ASCENDING SNO;
  SELECTION IS THRU S-S
  OWNER IDENTIFIED BY APPLICATION.
SET P-S;
  OWNER Parts;
          ORDER PERMANENT NEXT.
  MEMBER Stock;
          MANDATORY AUTOMATIC LINKED TO OWNER;
  KEY ASCENDING PNO;
  SELECTION IS THRU P-S
  OWNER IDENTIFIED BY APPLICATION.
```

Figure 8. Supplier-and-Parts Codasyl Schema Definition

2.4 Data Manipulation Languages

Data Manipulation Languages (DMLs) provide the means for querying and updating databases. Their comparison is facilitated through introducing the concepts of navigation and procedurality.

Querying or updating a database is ultimately accomplished through retrieving, modifying or adding record instances. The prerequisite is gaining access to the appropriate records (or logical location within the database if a record instance is to be inserted). This process is referred to as navigation.

Once appropriate access has been gained, i.e., an access path to the required logical location within the database has been constructed, additional actions, such as applying a predicate against the record instances, must be taken. Procedurality refers to the extent to which the user must specify how these actions are to be carried out rather than just specifying the predicate.

As an illustration of the various ways of gaining access supported by the different DBMSs, let us try, in our Supplier-and-Parts database, to locate the Supplier names and the names of the parts they supply. This requires accessing each Supplier record occurrence together with all its associated Stock records.

In relational calculus, this is done by first joining the Supplier and Stock tables on the common column SNO, then selecting the appropriate columns from this combined table, as illustrated below with MRDS DML [Honeywell 78].

```
"-range (s Supplier) (t Stock)
 -select (s.SNAME t.SNO t.PNO t.QUANT)
 -where (s.SNO = t.SNO)"
```

In relational algebra systems, we perform essentially the same operations, but in two separate steps (shown below in MULTICS Relational Data Management System (RDMS` DML [MIT 77]).

```
compose Supplier Stock -name Temp
project Temp SNAME SNO PNO QUANT -name Result
```

For hierarchical systems, this requires a sequential scan forward from the start of the database for all Supplier and all Stock segments as illustrated below for IBM's IMS [IBM 76].

```
     GU Supplier
NEXT GN Stock
     go to NEXT
```

The same kind of navigation is needed for DBTG systems: we traverse the Supplier record occurrences (by sequencing through the set A-S), and then traverse the occurrences of its member Stock records. A sample MIDS DML [Honeywell 77] sequence for this query is shown below (dbi is the database index and dbc is the status code returned by MIDS).

```
call dml_$find(dbi,"-first -path A-S",Supplier,dbc);
loop1:
call dml_$get(dbi,SNAME,dbc);
if (dbc=record_not_found) then goto exit;
call dml_$find(dbi,"-first -path S-S",Stock,dbc);
   loop2:
   if (dbc=record_not_found) then do;
      call dml_$find(dbi,"-first -path A-S",Supplier,dbc);
      goto loop1;
      end;
      else call dml_$get(dbi,SNO,PNO,QUANT,dbc);
   call dml_$find(dbi,"-next -path S-S",Stock,dbc);
   goto loop2;
exit: .
      .
      .
```

Intuitively, the user expends the least effort when the DML does not require user specification of the access path and, moreover, only requires that the user specify the appropriate predicates rather than, in effect, write a program for their implementation. Relational calculus languages such as SQL [Chamberlin 76] (cf. 3.4) only require a minimum of navigation (specifying the JOINs to be used in interrelating tables) and are nonprocedural. In contrast, the Codasyl DBTG DML is highly navigational and very procedural as illustrated above.

Generally, more complex data structures require more navigation. Since multiple access paths to a given record can exist, automating this navigation process proves relatively complex. Thus, high level DMLs have been considered more extensively in the context of the relational data model.

3. NETWORK VIRTUAL DATA MANAGERS

Currently, programs or users requiring access to a collection of remote, network accessible, heterogeneous DBMSs must formulate their queries or updates, hereafter termed requests, using the schemas and DMLs of the accessed DBMSs. Generally, the learning requirement implicit in this approach is sufficient to deter all but the most resolute user.

An alternative is to provide a common network-wide virtual view of data together with the appropriate DML. Through developing the required translation technology, requests expressed via this common view can be translated to those appropriate to the system being accessed. This provides a counterpart for multiple remote DBMSs of the functionality provided by other protocols for transferring files and executing distributed jobs.

Protocols are typically specified in terms of the services provided, the lower level services which are assumed, and the message exchange protocol governing communication between peer processes on different systems [Sunshine 80]. Protocols providing a common means for querying and updating remote heterogeneous DBMSs should be specified in the same way. However, the required translation technology must be substantially more sophisticated. Defining a uniform, network-wide view of data also requires considerable care in its specification and implementation.

There are two different approaches to assisting the network database accessor: i) requiring that the user have appropriate knowledge of each of the accessed systems, i.e., providing no assistance at all, and ii) forcing a migration from the collection of presently existing DBMSs to a distributed DBMS (DDBMS).

A DDBMS may be the most desirable solution. However, the cost of a forced migration to a DDBMS is substantial, since it requires database conversion--an expensive and only partially automated process. Further, if different organizational jurisdictions are involved, significant management participation may be required. The resulting implicit high cost of sharing may be sufficient to deter acceptance of DDBMS technology on purely organizational grounds. Since data structures must be tailored to support the differing requirements of two organizational units, the technical issue of achieving acceptable performance may also arise.

An intermediate alternative assists the network accessor by providing a uniform virtual view of data. This assistance is provided by a Network Virtual Data Manager (NVDM) which, logically, is intermediate between the terminal user or program requiring database access and the target DBMSs as illustrated in figure 9.

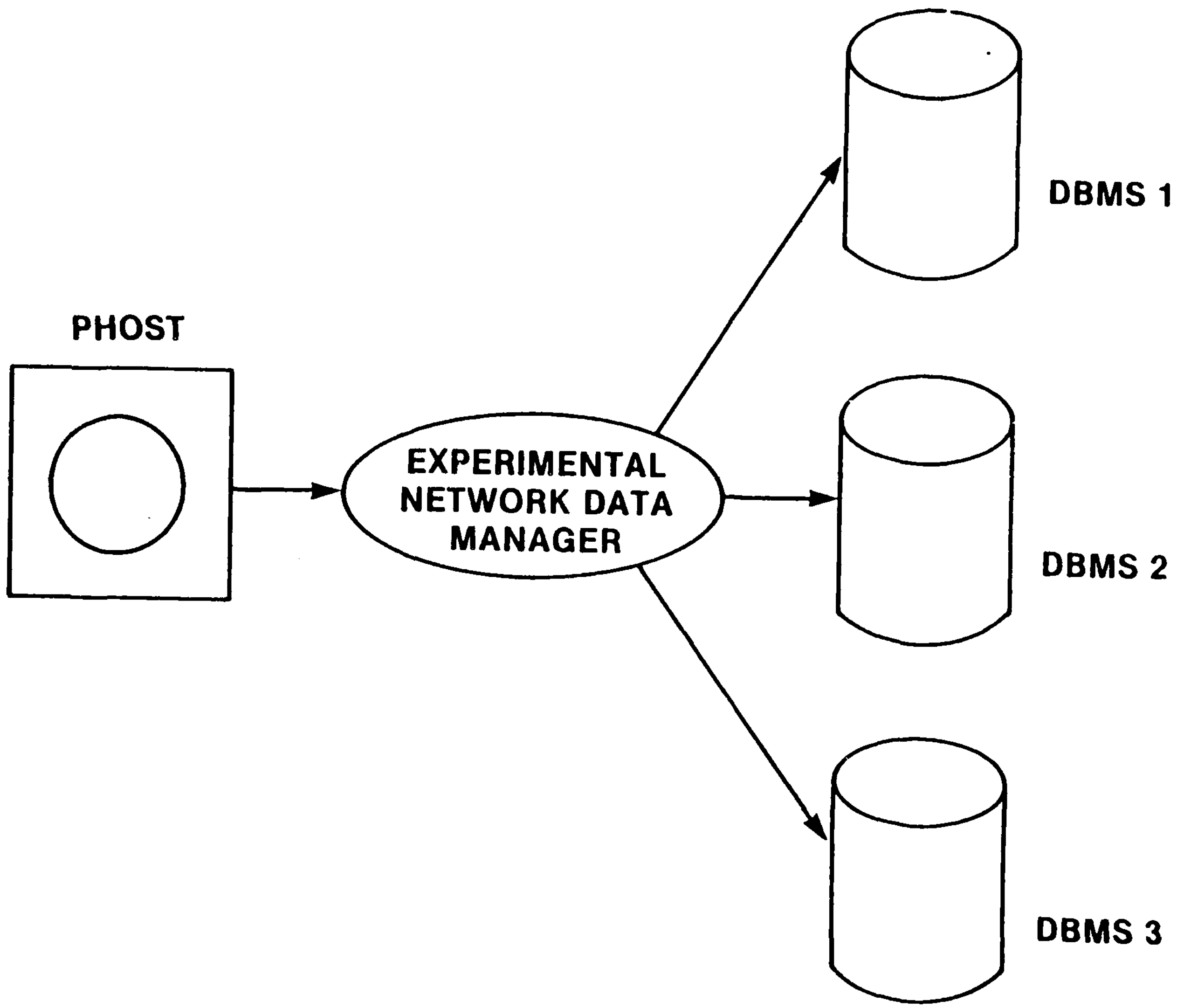


FIGURE 9. LOGICAL VIEW OF AN NVDM

The NVDM permits users to express requests in a single DML against a global schema describing a virtual network-wide view of data. Requests are decomposed into subrequests appropriate to the individual target systems. These subrequests are translated to the DML used by the target systems, processed, and the results aggregated and returned to the issuer of the request.

The advantages of having an NVDM include: reduced training costs; simpler incorporation of new DBMSs within the existing environment, i.e., ease of extensibility; minimization of the need for implementing DBMSs spanning organizational boundaries; and simplifying DBMS implementation through allowing data structures to be tailored to the needs of the predominant user body. The disadvantage is the need for an appropriate translation technology. Evidently, the advantages should outweigh the disadvantages if NVDMs are to experience operational use.

Interest in NVDMs originated from the need to provide a common view of data managed by independent DBMSs. The term "independent" reflects the assumption that the entities described by two different DBMSs are logically different. For example, if two different DBMSs support warehouse inventory control systems, both may contain "widgets" but the "widgets" in one are logically distinct from the "widgets" in the other.

The view of data presented by an NVDM can be made formally equivalent to that presented by a DDBMS by augmenting each table with an additional LOCATION attribute. The resulting schema together with the NVDM DML is equivalent to that provided by a distributed DBMS with the associated DML. It follows that from the user's viewpoint, there is no difference between an NVDM and a DDBMS.

NVDMs and DDBMSs differ substantially in their implementation. The major characteristic of the difference is that the NVDM is, in effect, a second level manager of individual DBMSs. (Network operating systems Kimbleton 78 are another interesting example of a comparable second level manager.) Thus, the NVDM must perform translation between the virtual view presented to the user and the view (global schema) supported by the individual DBMSs. Moreover, the level of functionality provided by the individual DBMSs may differ; such differences must be compensated for by the NVDM. Finally, the NVDM does not control resources; rather their control is entrusted to others (local DBMS management) and the NVDM, instead, occupies the position of a privileged user.

This section establishes major issues and alternatives in supporting a uniform network-wide view of data. The following section discusses its implementation. Together, the basic structure of a Network Virtual Data Manager (NVDM) is established. Although the user view is similar to that of a Distributed Database Management System (DDBMS), there are essential differences as discussed in 3.2 below. A description of an experimental implementation of an NVDM is presented in section 3.4.

3.1 NVDM Desirability and Structure

We doubt the desirability of an NVDM for highly repetitive or high bandwidth applications. However, information processing applications can be divided into three categories [Anthony 65]: operational control, managerial control, and strategic planning. As one passes from operational control to strategic planning, both the bandwidth of the application and the predictability of the accessed data elements decreases.

Evaluating the two factors of predictability and bandwidth, yields figure 10 giving our estimate of the relative utility of a Network Virtual Data Manager.

Access Support Feasibility

Resource Utilization \ Predictability	Low	High
Low	Yes	?
High	?	No

FIGURE 10. NVDM APPROPRIATENESS

Thus, low bandwidth and unpredictable applications are appropriate, while high bandwidth, highly predictable applications are better served by using the DML of the target DBMS. For the two boxes containing question marks, the relative desirability of the common network viewpoint is likely to depend upon NVDM ease of use and efficiency. Evaluation of these performance measures requires a significant amount of operational experience.

In evaluating NVDM desirability, two major factors should be considered. The first is the value of having the system be able to respond to queries and updates related to strategic planning via an integrated, on-line database access approach. The time savings can potentially involve orders of magnitude since the need for learning about the remote DBMS is eliminated, together with the likely requirement for active intervention by a trained programmer.

The second observation is that heterogeneity is ubiquitous. This reflects the fact that DBMS heterogeneity can be caused by: i) data model differences, ii) differences in data semantics, e.g. data structure differences, iii) DBMS functional differences, iv) DML differences, and v) computer system differences. Thus, an NVDM would probably still be necessary even if management required all sites to use a common DBMS package implemented on compatible systems.

Three key issues must be resolved in structuring a Network Virtual Data Manager. The first is the user's view of data, which is defined by the selected data model, the resulting virtual data structures and the DML available for manipulating these data structures. The second is the translation technology required to map operations expressed against this virtual view of data into those appropriate to the target systems. The third is the distribution of the translation process. Discussing these issues requires a clearer understanding of the sequence of NVDM processing operations which will now be described.

User—→NVDM

SEND	request expressed in a common, network-wide data manipulation language to the NVDM
DECOMPOSE	request into subrequests appropriate to individual target systems
TRANSLATE	subrequests from NVDM DML to DML used by target systems

NVDM—→Target DBMSs

SEND	translated requests to the target DBMSs
PROCESS	subrequests at target DBMSs; if request corresponds to query, place resulting record(s) in a buffer
TRANSLATE	buffer contents into a common network

representation used by all participating systems (Note that this use of the term translate differs substantially from the preceding. The preceding requires translation of a Data Manipulation Language while this and the following use involve translation of data. We have avoided using separate terms to conform with existing terminology. The precise type of translation required will be clear from context.)

Target DBMSs--►NVDM

TRANSMIT	translated records to NVDM for aggregation; reexpress results using common network-wide representation of data; and transmit to the requesting site

NVDM--►User

TRANSLATE	buffer contents at requesting site from common network representation to the site-specific representation
RETURN	result to the REQUESTing process

Issuing the SEND command assumes an NVDM schema and DML together with an appropriate supporting environment. These issues are discussed at length in the remainder of this section. Translating from this global view requires a translation technology differing substantially from that usually covered under the term data translation [SDDTTG 76]; one approach to its realization is described in section 4.

PROCESSing is assumed to be the province of the accessed DBMSs. TRANSLATEing the result requires a means for preserving the meaning of structured data being transmitted between heterogeneous systems. This problem is structured in section 5; a particular implementation is also discussed.

TRANSMITing data between systems requires a protocol. The requirements for a data transfer protocol are also discussed in section 5.

TRANSLATEing results from a common network representation at the requesting site is the inverse of translating partial results to a common network representation at the site where a portion of the DBMS query is processed.

RETURN of results to the requesting process is a function of the manner in which interprocess communication is implemented within a system and will not be explicitly addressed in this report.

Having described the entities involved in providing a common network wide view of data together with the required processing support, we now establish a framework for understanding key issues in constructing this common network wide view.

3.2 <u>Constructing the Network Wide View of Data</u>

The virtual NVDM view of data consists of a Network Virtual Schema (NVS) and a Network Virtual DML (NVDML) for supporting the specification of and interaction with the NVS. It differs from that of the local DBMS in data structures and data models, DMLs and coverage. The first two differences are evident; the third reflects the likelihood that local DBMS management may limit the data accessible by the network user to a subset of that actually managed by the local DBMS.

Selecting a data model and designing data structures should be based on user capabilities and needs. We assume that users: a) are responding to strategic planning and managerial control objectives, b) have limited systems knowledge, and c) are goal oriented.

Item (a) implies a user need for potential access to all of the NVDM accessible data. It also implies unpredictable, nonrepetitive requests. Thus, conceptual simplicity must be gained via representational clarity rather than through custom tailored views (external schemas) or abstract data types [Weber 78].

Limited knowledge coupled with goal orientation implies strong user interest in problem solving and a corresponding lack of interest in learning system intricacies. If an NVDM is not simple to use, it won't be used. Simplicity is dependent upon: i) the underlying data model which is being used, ii) the data structures based on this data model, and

iii) the DML available for manipulating these data structures.

3.2.1 Selecting a Data Model

Two criteria for selecting a data model are ease of use vs., perhaps, efficiency, and use of an existing rather than a proposed data model.

Given our assumptions about the user, ease of use dominates efficiency considerations. The virtual view of data which the NVDM presents has two important implications. The first is the effective elimination of the utility of performance tuning based on the anticipated nature of the user's requests. The second is that system maintained record type interrelationships--such as those presented by networks and hierarchies--are neither feasible nor useful. Thus, the decision to use a network or hierarchy as the basic data model must be based on the decision that the corresponding information structures are simpler to understand. Given the complexity of data likely to be supported by the global schema, we doubt that such a conclusion could be supported.

Existing data models are primarily concerned with the representation of records and their interrelationships. Proposed data models include semantic considerations. Although evaluating the relative merits of proposed data models promises to be interesting, from a representational viewpoint they seem to be primarily based on: defining things (entities), establishing attributes of things, grouping things into classes, establishing attributes of classes, and defining interrelationships between classes [McLeod 79].

While representing data via records has well established limitations [Kent 79], it is nevertheless satisfactory in many, if not most, instances. Since record interrelationships can be viewed as the province of data semantics, using tables as the basic data representation technique provides a reasonable basis for beginning the study of translation issues. (From the viewpoint of the translator, it makes little difference whether the "user" or the "system" specifies record-type interrelationships.) Many data semantics issues, as we discuss later, can be studied and implemented separately.

Given that: i) records are the basis for data representation, ii) the virtual view of data precludes the utility of system-maintained interrelationships, and iii) semantically meaningful interrelationships can be added as another layer (filter) as discussed in section 3.3, using a relational data model as the basic point of departure is reasonable.

3.2.2 Choosing Data Structures

Conceptually, the information contained within a database can be divided into two basic categories: information describing entities or things and information describing interrelationships among entities, other interrelationships or things. In the simple schema presented earlier, parts and suppliers are examples of entities whereas the information describing the number of parts procured from a given supplier is an example of an interrelationship.

Many database retrievals can be conceptually structured as determining attribute values of aggregates, e.g., average attribute values corresponding to a certain interrelationship among entities. For example, one might wish to determine the suppliers of red parts. This interrelationship is not explicitly contained within the schema and must therefore be derived.

Deriving complex interrelationships requires additional effort on the part of the user and provides an increased opportunity for error. Two alternatives exist: incorporation of (some) interrelationships within the schema, and duplicating certain data. The Entity-Relationship model developed by Chen [Chen 76] adopts the first approach for table-based representations of data. The utility of the second approach has also been argued convincingly [Hammer 79]. For an NVDM, these approaches are essentially equivalent, since the view of data is virtual.

The preceding suggests that the tables presenting the network user's (virtual) view of data should be logically structured into information describing entities and their interrelationships. Experience shows that such divisions are dependent on the viewpoint of the user [Hammer 79]. This poses a major problem in constructing database schemas. Fortunately, for an NVDM, supporting alternative views at the global schema level is feasible, since all views are virtual. This virtual view also permits incorporation of additional semantics with minimal effort; the prerequisite is their definition, representation, and expression via the DML.

3.2.3 Selecting a DML

Given a table-based view of data, two major DML alternatives exist. The first corresponds to the relational algebra and requires that the user explicitly issue commands for performing table JOINs, applying a predicate against the result, and selecting the necessary columns. Although this approach is flexible and, thereby, permits potentially greater efficiency in performing retrievals or updates, its procedurality is a disadvantage for the user population identified earlier. Interestingly, this procedurality is also technically disadvantageous because it is in conflict with the transaction processing approach deemed desirable by many [Gray 78].

Conceptually, the information contained within a database can be divided into two basic categories: information describing entities or things and information describing interrelationships among entities, other interrelationships or things. In the simple schema presented earlier, parts and suppliers are examples of entities whereas the information describing the numoer of parts procured from a given supplier is an example of an interrelationship.

Many database retrievals can be conceptually structured as determining attribute values of aggregates, e.g., average attribute values corresponding to a certain interrelationship among entities. For example, one might wish to determine the suppliers of red parts. This interrelationship is not explicitly contained within the schema and must therefore be derived.

Deriving complex interrelationships requires additional effort on the part of the user and provides an increased opportunity for error. Two alternatives exist: incorporation of (some) interrelationships within the schema, and duplicating certain data. The Entity-Relationship model developed by Chen [Chen 76] adopts the first approach for table-based representations of data. The utility of the second approach has also been argued convincingly [Hammer 79]. For an NVDM, these approaches are essentially equivalent, since the view of data is virtual.

The preceding suggests that the tables presenting the network user's (virtual) view of data should be logically structured into information describing entities and their interrelationships. Experience shows that such divisions are dependent on the viewpoint of the user [Hammer 79]. This poses a major problem in constructing database schemas. Fortunately, for an NVDM, supporting alternative views at the global schema level is feasible, since all views are virtual. This virtual view also permits incorporation of additional semantics with minimal effort; the prerequisite is their definition, representation, and expression via the DML.

3.2.3 Selecting a DML

Given a table-based view of data, two major DML alternatives exist. The first corresponds to the relational algebra and requires that the user explicitly issue commands for performing table JOINs, applying a predicate against the result, and selecting the necessary columns. Although this approach is flexible and, thereby, permits potentially greater efficiency in performing retrievals or updates, its procedurality is a disadvantage for the user population identified earlier. Interestingly, this procedurality is also technically disadvantageous because it is in conflict with the transaction processing approach deemed desirable by many [Gray 78].

The alternative DML type corresponds to the relational calculus and stresses specifying what data is to be accessed (rather than how) and the predicate used for selection. This is simpler for the user; it is also much more efficient for the NVDM, since it permits a transaction-based implementation approach. Thus, a single DML request is likely to correspond to the user's perceived request rather than just a fragment. This request can be translated and the resulting aggregate information transmitted to the target system where it is processed as an entity (cf. the discussion of envelopes below).

3.2.4 Performance Considerations

The NVDM performance is driven by four basic factors: i) data structure effects, ii) the performance of the translation process, iii) target DBMS performance, and iv) distribution effects.

Data structure effects reflect the possibility that JOINs which seem natural for the network user may be poorly supported by the target systems. Section 4 contains an extensive discussion of other data structure effects.

A natural and interesting metric for measuring the performance of the translation process would compare the performance of the code resulting from translation with that produced by a trained user of a given target system.

Unfortunately, such comparisons will only be meaningful after NVDMs have reached some initial stage of maturity. Current efforts such as those discussed in section 4 are directed toward establishing feasibility and key problem components rather than optimization.

The performance of the target system in processing a query or request is presumably fixed. However, knowledge of optimization information maintained by the target system can permit generation of equivalent queries or updates having varying performance levels.

Distribution effects reflect the degree of interaction required in processing a given request by modules located on different systems. We believe such interaction should be minimized and are in agreement with Gray's observation that a transaction-oriented viewpoint is appropriate. Attaining this objective when the target system uses a low level navigational and procedural DML such as the Codasyl DBTG DML requires establishing somewhat sophisticated server functions at the target DBMSs sufficient to support execution of an entire query.

3.3 Data Integrity

Table-based data structures define the NVDM representation of data. They do not provide substantial or strong guarantees that the network user's queries or updates will be meaningful and, in the case of updates, that incorrect data will not be entered. To allay fears in the minds of local DBMS management, it is highly desirable that some guarantees be provided about limiting the accessor's view to that appropriate to the access rights which are possessed.

Integrity requires filtering DML requests. Three levels are identifiable: access controls, maintaining meaning, and simplifying specification. Access controls ensure that the access rights of the user are appropriate to the access requirements of the data specified in the request. Maintaining meaning encompasses both assuring semantic integrity of updates as well as ensuring that queries are semantically meaningful. Specification simplification seeks to simplify stating requests through minimizing the amount of information which must be explicitly stated by the user.

3.3.1 Controlling Access

Access controls provide the basic mechanism defining who can access a given data element and, for each such accessor, the permitted types of access. We distinguish between access control primitives which are provided and the guarantees that they work exactly as specified, cannot be circumvented, and correctly implement an articulated security policy. Assessing these latter issues is a key component in determining the security of a system and will not be discussed further. Thus, our concern is with the nature of the mechanisms which can be provided to control access rather than with guarantees about the correctness of their operation.

Two types of access controls are nondiscretionary and discretionary. Nondiscretionary access controls provide the means for imposing organizational constraints on data sharing. Typically, they are implemented as a collection of security levels, e.g., CONFIDENTIAL, SECRET, and TOP SECRET, together with compartments within levels, e.g., NATO and NUCLEAR. The security policy controlling their use has two major assertions; users with the same security classification can always see the same information, and a user can always access information whose access requirements are dominated by his/her access authorization. Thus, a user with a classification (TOP SECRET, NATO) could access information classified (SECRET, NATO).

Discretionary access controls provide the mechanism for supporting user-controlled sharing of resources. In contrast to nondiscretionary controls, there is no attempt to conform to any given security policy; instead, the objective is to provide a means for enabling users to grant access among themselves as they see appropriate. SYSTEM R [Astrahan 76] provides a rather sophisticated collection of database-related discretionary access control mechanisms. An individual user can grant a wide range of access options to another user (READ, INSERT, DELETE and UPDATE tuples of a relation or, perhaps, only certain columns, and DROP entire relations). In addition, the user can also grant the GRANT right. This permits the grantee to further grant access rights which have been received. This rather fine-grained mechanism permits substantially better control over discretionary access than the password based, file oriented protection mechanisms common to many of the existing DBMSs.

The desirability of supporting both discretionary and nondiscretionary access controls within a given system is evident. Their joint support requires a mediator for detecting and resolving conflicts between them. The mediator must ensure that discretionary grants are never in conflict with nondiscretionary controls.

The second function of the mediator is ensuring the star property. This property states that a user with a given level of security classification can write, e.g., append, data to information at a higher classification and can read data of a lower classification. The star property is the basic consistency requirement which must be adhered to when the user is accessing data at a different security classification than his own. Through its enforcement, one can ensure that the security requirements reflected in nondiscretionary access controls are not violated [Denning 76].

3.3.2 Maintaining Meaning

Intuitively, database semantic integrity is concerned with assuring "clean" data, i.e., assuring that the data within the database is an accurate model of the organization. Traditionally, this is achieved by beginning with a semantically correct database, and then assuring that subsequent updates are also semantically correct.

Accomplishing the objective of semantic integrity is a complex undertaking even within the confines of an individual database. This reflects the implicit requirement that the organizational meaning of the database be specified.

Two major approaches to specifying database semantics exist. The more common is based on the use of assertions specifying certain interrelationships within the data managed by the DBMS. This approach has the advantage of simplicity and incremental implementability. The disadvantage is the infeasibility of determining when the collection of assertions is complete. This approach has been implemented in varying degrees by several projects [Griffiths 76], [Stonebraker 76].

The alternative approach is based on a significant extension of the concept of data type [Brodie 78]. The formal specification of these types, together with their interrelationships, allows one to develop a complete specification in a highly stylized way. However, incremental specification is precluded. Currently, this approach has not yet been implemented.

The preceding approaches to semantic integrity are reasonable in the context of an individual DBMS. However, if semantic integrity is to be provided at the NVDM level, several differences exist. The most important is the fact that the NVDM does not have control over the data. Additionally, there is no way of guaranteeing that the actual databases satisfy any set of semantic integrity assertions. Moreover, semantic integrity assertions for two different databases supported by the NVDM may well be in conflict and yet be correct with respect to the system for which they have been asserted. For example, one system may require SALARY less than or equal $30,000 while another may require SALARY less than or equal $20,000. The SALARY of an individual making $25,000 would be consistent with the semantic integrity constraints of one DBMS and not with another.

A second difference in NVDM supported semantic integrity arises from the likely limitation of network access to a subset of the data managed by an individual DBMS. As a result, there is no way of checking semantic integrity assertions spanning data accessible to the NVDM together with data which is not accessible to the NVDM.

The preceding suggests that semantic integrity in the NVDM context differs significantly from individual DBMS semantic integrity and, additionally, will be fundamentally less powerful. If this is unacceptable, an alternative is to have all semantic integrity related issues handled by the local DBMSs. Thus, semantic integrity assertions would not be checked until the translated query has arrived at the local DBMS. Unfortunately, given the present incomplete state of understanding of semantic integrity systems coupled with the fact that they do not exist as products, this approach is practically of little value.

A second approach would be to map all semantic integrity assertions from the local DBMS to the Network Data Manager while taking into account the differences in data models which may exist. Reasons cited in the first approach, coupled with the likelihood of missing data and the consequent impossibility of checking semantic integrity assertions involving such missing data render this approach unacceptable.

The third approach, which is being adopted in the context of the Experimental Network Data Manager [Kimbleton 79] described below, discards completeness as a major consideration and broadens the range of semantic integrity considerations to include both queries and updates instead of only updates. The extension to include queries reflects the desire to check that queries are basically meaningful.

For example, a violation would be noted if one attempted to compare SNAME to PNAME. That is, strong domain typing is used to ensure semantically meaningful queries.

To assist in assuring the semantic integrity of updates, assertions and some data type extensions are being supported. Conceptually, these assertions can be divided into five categories: value assertions, row assertions, column assertions, intrarelation assertions, and interrelation assertions. The cost of checking the last two types of assertions is likely to be high. Accordingly, such assertions should only be made if absolutely necessary. The second principle to be followed is that whenever possible, assertion checking should minimize data retrieval requirements.

In summary, semantic integrity is likely to be an even more important concept for an NVDM than for an individual DBMS. However, significant differences exist with the result that obtaining a complete set of semantic integrity assertions at the Network Data Manager level will usually be practically impossible and may well be theoretically impossible.

3.3.3 Simplifying Specification

The virtual view of data provided by NVDMs provides a very reasonable testbed for evaluating new, and more powerful DMLs and data models. Thus, incorporation of information describing classes, their attributes and interrelationships, and implementation of DMLs for manipulating this information is substantially simpler with an NVDM. This reflects the fact that retrieval issues can be minimized and consequently, attention can be focused on representational and manipulative issues. This facilitates studying alternative network user views of data.

3.4 XNDM--An Experimental Network Data Manager

To investigate the feasibility of constructing an NVDM, an Experimental Network Data Manager has been preliminarily studied at the National Bureau of Standards. The following presents a short overview of XNDM. A more detailed description is contained in section 4.6 and in a previous paper by Kimbleton [Kimbleton 79].

3.4.1 XNDM User Interface

The XNDM user interface consists of a Network Virtual Schema based on tables, together with a Network Virtual Data Language. The DML portion of the language consists of XNQL and XNUL. XNQL is the abbreviation for Experimental Network Query Language while XNUL abbreviates Experimental Network Update Language. This division has been established because of the significantly differing requirements implicit in

C1	Select
C2	Select . . . Where
C3	Aggregation
C4	Partition
C5	Set Operations
C6	Composition

FIGURE 11. XNQL PRIMITIVE CATEGORIES

supporting these two components. (The authors have also noticed that local DBMS management seems much more willing to support remote querying than remote updating!)

The XNDM Data Language is modeled after SQL [Chamberlin 76]. The primary reason for selecting SQL was its orientation toward a transaction processing style of interaction between the user and the DBMS. Additional factors supporting this decision were: i) its relatively complete, publicly available description, ii) its table orientation, iii) its analysis in terms of human factors, and iv) the likelihood that the XNDM supported local DBMSs would not use SQL since IBM systems are of limited accessibility on the ARPANET.

Figure 11 lists the six major classes of XNQL primitives.

Classes 1 and 2 provide the basic addressability required to support primitives of classes 3-5. Class 6, composition, supports the ability to nest queries and, thereby, facilitates a transaction processing orientation. A more detailed discussion is given by Wang [Wang 80].

3.4.2 XNDM Translation

Currently, the feasibility of the required translation technology has been demonstrated. Development of an appropriate structure relating access path specification and translation is under investigation. Section 4 contains a detailed description of our interim understanding of this problem.

3.4.3 XNDM Distribution

The XNDM implementation is distributed between the translator (resident on a PDP-11/45 as discussed below) and server modules, termed envelopes, resident on each of the target systems. The envelope serves two primary functions: i) presentation of a uniform DBMS functionality within a given data model, and ii) support of a transaction processing oriented viewpoint. Item (ii) is particularly important, since it serves to minimize network bandwidth requirements as well as total delay.

3.4.4 XNDM Prototype Implementation

The current prototype XNDM implementation uses three ARPANET accessible computers. The host computer can be any ARPANET accessible host; currently it includes a PDP-11/45 running UNIX (UNIX is a trademark of Bell Laboratories, Inc.) [Ritchie 74], a Honeywell 6180 running Multics and a PDP-10 running TOPS-10. The two DBMSs are the Honeywell Multics Relational Database System (MRDS) [Honeywell 78] and Honeywell Integrated Data Store (IDS) [Honeywell 71] implemented on the RADC Multics system. Translation is supported on a PDP-11/45 running the UNIX operating system.

4. TRANSLATION TECHNOLOGY

This section considers the construction of the query management portion of NVDMs. It describes the functional nature of the task involved in the building of such systems and provides an analysis of the task. We then set forth a set of design aims for the construction of NVDMs based upon studies of the nature of the task and the operational characteristics of the application environment. A general solution approach is also outlined. A substantial fraction of the section is devoted to a detailed examination of the XNQL translator as a concrete illustration of the proposed solution methodology.

The XNDM is a network virtual database management system, that is, it is a facility intended to support the use of remote DBMS capabilities in a network-wide, unified manner. Thus, it provides a Network Virtual Schema (NVS), and a data manipulation language, XNQL, for the user to express data handling procedures. To the user, XNDM provides all the services that a conventional (single-host) DBMS provides.

It is important to emphasize that although XNDM appears operationally as a state-of-the-art database manager, it is not based on, nor does it involve, the writing of a new database management system. Rather, it is built by using the existing facilities for database accessing and manipulation supplied by the individual remote DBMSs.

The above discussion emphasizes that the environment in which XNDM is run is that provided by the host database management systems (sometimes augmented in certain host DBMS deficient areas, or molded into a form more appropriate for incorporation into an NVDM). Therefore, the essential function of XNDM is the coordination of the different individual DBMS operations, not the provision and execution of these operations.

The heart of XNDM is the query language XNQL, which provides a set of commands allowing interaction with the DBMSs of other network hosts. Since we rely on the existing individual DBMSs for access and retrieval of stored data, the task of query evaluation reduces to that of translating user inputs in the form of XNQL statements (i.e., a sequence of XNQL operations on the underlying NVS tables) into remote DBMS operations upon the data structures maintained by the individual systems.

4.1 Nature of the Translation Function

Before turning to the subject matter of this section, it is worthwhile to characterize the functions of an NVDM in (semi-)formal terms so that we shall have a more rigorous framework on which to base later discussions. (Eventually we hope to express the problem in an exact mathematical form so we can apply mathematical techniques to deduce various

AD-A102 776 NATIONAL BUREAU OF STANDARDS WASHINGTON D C INST FOR--ETC F/G 9/2
XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER.(U)
JUN 81 S R KIMBLETON, P S WANG, E N FONG F30602-77-C-0066
UNCLASSIFIED RADC-TR-81-123 NL

2 OF 5
AD
A102776

properties for the system, e.g., correctness. Right now we shall not attempt to make the discussion precise.)

4.1.1 A Formal Definition

Following Mealy [Mealy 67], we say that a database system is a representation of some aspect of the real world, expressed in terms of a set of logical data structures (i.e., a "data model"), a set of instructions for accessing and manipulating these data structures, and a set of values constituting the inputs to and outputs from the system.

We characterize a system by the relation between inputs and outputs which it determines, and also by the path by which we get from the input to the output. Our concern with the path is due to our desire to distinguish between different ways of performing the same database operation. The path information tells us both the instruction execution sequence and the state of the logical data structures which are generated along our way from the input to the output.

Consider a database system in the above sense, that is, a triple:

$$S = (E, A, V),$$

where E is a set of record types, V a set of values, A a set of "access functions" whose members are maps of the form:

$$\underline{t} : E \rightarrow V.$$

(Here, as in the entire section which follows, we are confining ourselves to a much smaller context than is usually found in discussions on database systems. Specifically, we are only addressing the database retrieval, or querying aspect of the system. For general database management systems, a full set of database manipulation functions should be used in place of the access function A in the above definition for S.)

In terms of the above model, a query is defined as a composition of database access operations. Its semantics are determined by the observable sequence of data structure state transformations generated by the query during execution.

4.1.2 Access Functions

Access functions provide the mechanisms for probing and traversing the database. Some of these allow us to locate data elements or records directly, e.g., keyed and indexed retrievals; others allow us to determine the location of data records through associations, e.g., retrieval by links.

As an example of an access function, the XNQL selection operation

```
"select x.QUANT from Stock x"
```

retrieves the collection of quantity values from the value set consisting of the QUANT column of the Stock table. The Codasyl DBTG statement

FIND FIRST STOCK WITHIN S-S.

has as its value set V the set of all records of type Stock. This is an instance of a "structural map" Mealy 67 , which is a map of the form

$\underline{t} : E \rightarrow E$.

Formulated in this way, any retrieval operation can be decomposed into a record-type mapping (i.e., Mealy's "structural data map") followed by a non-record-type mapping, that is, a mapping whose result is a collection of data values rather than record-types, or, in other words, elements of the set

$W = (V - E)$.

For example, in the XNQL query

```
"select x.QUANT from Stock x, Supplier y
   where x.SNO = y.SNO and y.SNAME = "Smith"",
```

there is an initial mapping which, given the value "Smith", yields all instances of the record(tuple)-type Supplier. (This can be thought of as a "direct" or "keyed" access.) It is followed by another mapping which takes tuples from the Supplier table and gives us a subset of the tuples from the Stock table (a "linked" access). A final mapping (a "projection") then yields a collection of quantity values from these Stock tuples.

4.1.3 Equivalence Criterion

In terms of the above framework, the XNDM database system S, and the target host's database system S' are merely different representations of the same model of the real world. This requires that we can assign a unique object in S' to any object in S, and vice versa. In other words, there must exist a representation map, u which maps S into S':

$$\underline{u} : (E, A, V) \rightarrow (E', A', V'),$$

and $\underline{u}$ must be one-to-one and onto [Mealy 67]. Furthermore, in order to insure that every fact derivable from one system is also derivable from the other [McGee 74], we require that for every valid query q in S, the situation shown in the following diagram can be obtained.

$$\begin{tikzcd}
E \arrow[r, "q"] \arrow[d, shift left, "s^{-1}"] & W \\
E' \arrow[u, shift left, "s"] \arrow[r, "q'"] & W' \arrow[u, "i"']
\end{tikzcd}$$

FIGURE 12. EQUIVALENCE OF ALTERNATIVE MAPPING STRATEGIES

In the above diagram, $\underline{s}$ is the structural mapping process which allow us to derive the NVS tables from the individual remote databases:

$$\underline{s} : [E' \rightarrow E],$$

$\underline{i}$ is the identity map, and q' is the (sequence of) target commands corresponding to q. That is,

$$q' = \underline{v}\, q,$$

where $\underline{v}$ is the operational translation which establishes the correspondence between the XNQL operations and the individual DBMS data manipulation commands:

$$\underline{v} : [A \rightarrow A'] .$$

In other words, starting with an element of the allowable class of target data structures, e' "is a member of" E', one gets the same output values by applying the query q' as by applying $\underline{s}$ followed by q.

4.2 Translation Alternatives

The previous discussion clearly shows that there are two alternative approaches to the problem of XNDM-target DBMS translation. This subsection describes these two alternatives and briefly sketches reasons for choosing one over the other.

4.2.1 Data Structure Mapping vs. DML Translation

In the first approach, the desired results are obtained by transforming the target data structures into equivalent NVS representations, followed by the application of the XNQL query q on the NVS structures so generated. This way of viewing the problem emphasizes the structural aspects of the underlying databases, since translation is done by reorganizing the target data into NVS schemas. The later processing of the user query q can be performed in essentially the same manner as that for any conventional relational calculus system. (There exists a large body of literature on database structural mappings, see, for example, McGee [McGee 74], Navathe and Fry Navathe 76 and Biller [Biller 79]).

The second approach carries out the structural and query mappings (i.e., the s's and the q's) in the opposite order from that described above. Here we first establish the correspondence between q and a (sequence of) target DMLs (q'), then perform query evaluation by executing the translated query q' against the actual target data structures [Paolini 77].

4.2.2 Decision Rationale

The first approach is unattractive for two reasons. First, it requires a large amount of data to be transferred from remote hosts to the NVDM machine to materialize the (relevant parts of) NVS tables. Since data transmission between NVDM and target sites is via communication links, this process is usually slow, sometimes unreliable or insecure, and always expensive. Secondly, we are making very ineffective use of remote DBMS services by performing a large part of the query processing inside the NVDM, instead of delegating them to the remote host.

We are thus led to the adoption of the second approach. This requires a well designed XNQL-to-target DML translator in order to be able to handle a large number of diverse target systems on a machine with relatively modest computational power and memory capacity (a PDP-11/45 with 128K core memory). This topic will not be elaborated upon here, as a detailed discussion of the solution approach will be taken up in a later subsection.

4.3 The Query Translation Process--An Informal Description

The above characterizes the network query language translation problem as that of deriving the procedure v which maps the access functions A (on the class of data structures E) into a new set of access funtions A' (on a new class of data structures E').

A question arises here naturally: what are the basic functions such a procedure needs to perform to derive the functions A' from A? Before going into an in-depth analysis of the various problems in devising the actual mapping procedure, let us describe intuitively how such derivation can be carried out. We shall use Codasyl DBTG as our target DBMS, since it sheds more light on the subject than hierarchical or relational systems which are simpler and closer to the NVS.

4.3.1 Determination of Query Paths

A basic XNQL query retrieves specified data, the "selection list", from a database, possibly (but not necessarily) subject to some contraints - the "where clause". This must be transformed into a DBTG "query path", (a sequence of DBTG accessing primitives, i.e., FINDs) to access the target data.

Intuitively, the first step in XNQL translation is to identify the set of record-types that needs to be accessed. We begin by discovering the mapping between the set of XNQL retrieval items and constraint items and the DBTG data structures (records) of which those items are a part. Moreover, we note that, in general, the retrieval and the constraint items may not share common record-types. This means that we cannot access one record-type directly from the other, so we need to find some substructure which includes both the above sets of data items and which forms a "valid" underlying data structure for the specific XNQL retrieval request we are considering.

There are many possible definitions of what constitutes a "valid" substructure, none of which is completely satisfactory [Collmeyer 72]. For the sake of discussion, we shall adopt the very practical, but restrictive, approach of Lavallee et al. [Lavallee 72] and define it to be one which satisfies the criterion of a "valid covering". A covering is defined to be a connected subgraph of the data structure diagram for the database [Bachman 69] which includes both the retrieval and the constraint record types. A covering is valid if and only if it conforms to the specifications of a confluent hierarchy (see [Collmeyer 72] for detailed discussions and examples of confluent hierarchies).

Given a valid covering, we can approach the problem of query path determination in the following way. We start with any directly accessible record-type R (i.e., member of a DBTG singular set or one with LOCATION MODE CALC) and mark it as visited. Next we visit each of the unvisited record-types adjacent to R. Then unvisited records adjacent to these records are visited, and so on. The search terminates when no unvisited record-type can be reached from any of the visited ones.

This breadth-first search procedure [Horowitz 76] generates all possible paths for traversing the valid covering of the query. Usually, since more than one such path exists, we need another stage in the path formulation process -

selection of the "optimal" path, to which we shall now turn our attention.

4.3.2 Selection of An Optimal Path

The objective of this stage is to find the query path that uses the least amount of resources (cpu time and secondary storage accesses). For the sake of discussion, we equate this with the number of (expected) logical record accesses. To a first approximation, this problem is separable into two parts. The first part is minimization of the number of record-types in the query path. Next, we minimize the number of record instances which must be traversed. This can be done by identifying the position of the constraining records in the path, and choosing the one where these records are as near to the entry record as possible. (An alternative and simpler way to do this is as follows. In each step of our breadth-first search we pick, whenever possible, not an arbitrary adjacent record as prescribed in section 4.3.1, but one which is constrained to be our next node to visit.)

Of course, the above strategy has to be improved in many ways before it can be used in a real application. First, we have assumed that the relative effectiveness of different constraining records is the same, whereas in actuality, this is usually not the case. Second, traversal of different sections of the query path usually incurs different "costs". Therefore we need a more sophisticated way of estimating resource usage than a straightforward counting of the record instances traversed [Astrahan 76], [Yao 79].

4.3.3 Limitations

The above approach has serious difficulties so far as actual implementation is concerned. Its basic inadequacy lies in its essential appeal to our intuitive feeling of what constitutes a "valid" substructure for an XNQL query. It also depends critically on our ability to show that the "meaning" (i.e., the operational effects) of the source XNQL statement is preserved by the sequence of DBTG FINDs that we produce for traversing this substructure.

To justify this approach, we need to view the task facing the XNQL translator in a more detailed way. We shall consider the operational results of each of the XNQL constructs and find out how to achieve the same effects with DBTG DMLs. Based upon these observations, we then devise a set of rules that describe the sequences of DBTG DMLs as a function of the NVS-DBTG schema context. Such a set of rules forms the basis of the algorithm for the translator to determine, step-by-step, the "valid" target substructure and the "valid" traversal path. This will be the topic of the following subsection.

4.4 A Taxonomy of Major Translation Issues

It is time now to look at the translation problem in more general and detailed ways. The present subsection attempts to analyze the task situation. A discussion of the XNDM solution approach will be given in section 4.5.

The two basic issues involved in network virtual query translation are data schema and operation mapping. They correspond, roughly, to finding the valid covering data structure and generating the (optimal) query path, respectively, as described in the above subsection. These issues, and their inter-dependencies, will be dealt with individually in the following paragraphs.

4.4.1 Data Schema Mapping

Schema mapping provides for the connection between NVDM data structures and the various remote DBMS data structures. The most obvious such connection is that between the individual data items -- the first type of mapping.

It is necessary for a network virtual data manager to be provided a means for referencing data managed by remote DBMSs. Each DBMS has some internal scheme for naming data items, but these various schema are typically different and may even be incompatible. Since it is impractical (and in many cases, impossible) to impose a common internal data item naming scheme, a network virtual name space is used with a portion of the name space allocated to each remote DBMS. The relationship between the remote DBMS's internal name space and its portion of the virtual data item name space are used by the data mapper to transform references to NVS attributes into those for individual DBMS data elements.

The second type of schema mapping relates the NVS record-types (i.e., the XNDM tuple construct) to the record-types supported by the remote DBMSs. This implies that aggregation or splitting of tuples may be needed in order to map references to XNDM tables into those for the local DBMS record-types. Since almost all generalized database systems contain the basic data structure corresponding to (or closely parallelling) records (e.g., records or entries in the Codasyl DBTG proposal [Codasyl 73], segments for IMS [IBM 76], logical records for IDS [Honeywell 71], relations for INGRES [Stonebraker 76] and MRDS [Honeywell 78]), the practicality of the previous two schema mapping functions is common across all individual remote DBMSs.

A third type of mapping deals with the interrelationships between the different record-types. In contrast to intra-record structures, the common generalized DBMSs support widely different inter-record structures, as evidenced by the fact that almost every system has a different set of facilities for traversing the various record-types within its data space.

For example, relational calculus systems [Codd 71] allow requests for subsets of the data stored in the system to be made completely in terms of first-order predicate calculus operations on the entire collection of database entry types (relations), without specifying search procedures and paths to be followed. Codasyl DBTG systems, on the other hand, provide for a very complete specification of how the search is to be conducted in terms of the exact path to be followed. It is necessary to specify the location of the next record instance from the current cursor position within the database. Relational algebra systems [Codd 71] present yet a third way of database navigation. These systems require the user to formulate a sequence of unary and/or binary algebraic operations on the underlying database relations, thereby generating the desired data subsets in a stepwise manner.

Therefore, the makeup of the relevant "inter-record relationship mapper" depends heavily upon the individual local DBMS. Later in this section we shall discuss, in detail, the various issues involved in constructing such a mapper in the context of a specific system - XNDM.

4.4.2 Operation Mapping

Operation mapping translates the XNQL table operations, namely, column and row selection, aggregation, partition, set operation and various compositions of these operations, into the corresponding operations for handling target system database records and the logical interrelationships among these records.

There are four aspects to the general problem of mapping between the data manipulation operations of the various systems. The first problem we need to address is the difference in meaning (sometimes subtle, and always annoying) of primitive data manipulation operators as evidenced by the operational effects these primitives produce. A simple example of such a difference is the different degree of accuracy exhibited by the floating-point arithmetic operators for various machines [Neely 77]. Another example is the non-uniform treatment of duplicate results by the currently existing relational systems [Honeywell 78], [Chamberlin 76].

Moreover, different DMLs typically differ in the level of data aggregation allowable as a unit of manipulation [Stonebraker 75]. Some languages, notably that proposed by the Codasyl Data Base Task Group, dictate an instance-at-a-time query style. These languages contain constructs that return exactly one instance of a record each time it is called (e.g., the DBTG "find" statement). Hence, the user must execute a proper sequence of such commands in order to acquire, instance by instance, the needed information. XNQL, in contrast, is a "set-oriented" language, that is, its statements return information from the entire set of record instances satisfying the indicated condition specified in the command.

Another dimension in which query languages can vary is the degree of history sensitivity they exhibit Backus 78 . Some DMLs include the notion of storage, so that one statement can save information and thereby affect the behavior of subsequent statements.

For example, the definition of Codasyl DBTG DML Codasyl 73 is based upon the fundamental concept of currency, which constitutes the essential channel through which interactions between individual DMLs take place. With systems of this type, the user constructs the query a piece at a time by applying transformations on this storage space of currency indicators. This is why FIND is the most important statement for DBTG systems [Date 77]. It is logically required before each of the other statements (except STORE) so that these statements can have the right storage environment to operate in.

Languages like SQL or XNQL provide much more powerful ways for constructing complex queries. The primitives of the language operate on whole conceptual units of data (tables) and produce whole conceptual units, and we compose individual queries q and q' by first applying q and then using the result as an argument to q'. This property is very desirable, since by confining the operation of individual statements to only its arguments, we do not need to worry about hidden states (contents of currency indicators and temporary storage areas, and other side effects) and the complex transitions these states undergo.

Finally, perhaps the most important characterization of a database system concerns its mechanisms for expressing associations between the different record-types within the database. Some systems, among which the Codasyl DBTG [Codasyl 73] and the IBM Information Management System [IBM 76] are the best known examples, provide specific constructs (e.g., the DBTG "set" and the IMS parent-child relationship) for interrelating different record-types. These data structuring constructs are used by operators ("find next" and "find owner" for DBTG, and "get next" for IMS) for retrieving information contained in logically related, but different, data records.

Relational systems [Codd 70] choose, instead, to eliminate all "access path dependencies" from their data structures, and to express associations between record-types explicitly in the data element values contained within the record-types (relations). Prescriptions for interrelating record-types are supplied dynamically in the queries issued by the user in the form of join operations on two relations over a common ("join-able") column.

The entity-relationship systems [Chen 76] take the view that record-types should only be used to model real world entities. Associations among record-types should model real world relationships between these entities and are part of the meta-information the system maintains about the database. The logical analysis needed to resolve the interrelationships among various record-types is performed automatically by the system so the user need not express in detail how to "chain" through the different records [Shoshani 78].

4.5 Implementation Approach

We have analyzed the various problems a network virtual data manager has to solve. We would now like to provide an algorithm for the translation process. Not all issues involved in the building of an NVDM will be considered here, nor is every facet of the proposed solution unique or optimal. Our purpose is to present one prototype real implementation, namely, an experimental translator for our network virtual query language, XNQL. Another purpose of this subsection is to illuminate the many aspects of the problem of virtual DML translator construction based upon experiences gained from this experiment.

4.5.1 A Translator Family

Implementation of the XNQL translator is complicated by the fact that different target systems support different primitive operations and data structures; therefore, we need not a single translator but a family of translators. Two approaches to their realization can be identified: construction of a collection of source-target specific translators or, alternatively, construction of a single translator for the bulk of the translation process common to all individual DBMSs, together with post-processing modules for handling the target-specific portion of the translation process.

Construction of independent translators has the advantage that design unity and run-time efficiency is more achievable with a single translator for each target DBMS. However, an entire translator is needed to support each additional target. In the family approach, all the translators share a core design which defines the common (target-independent) part of the translator. Each new translator in the family is obtained by building the required target-oriented functionality on top of the basic design. Therefore, the bulk of the implementation effort is available across different target systems and new developments need not start from scratch.

4.5.2 Environmental Constraints

For the XNDM application environment, easy adaptability to different target DBMSs is of primary importance. The reason is that practical use in a production environment requires that an NVDM accomodate the various evolutionary modifications to which individual DBMSs are constantly subjected [Boehm 73].

Another design constraint is that the translator only act as a mediator between the XNDM view of data and that of the remote database managers and must, therefore, interact with the target DBMSs as a user of the system. The reason for the imposition of this condition is our basic assumption that the hardware, software and human resources needed for XNDM operation are both geographically and managerially dispersed, although connected through a network. Therefore, interfacing to remote facilities in any other manner would present difficulties, both operational and administrative, which would be very difficult, if not impossible, to solve.

4.5.3 Design Goals

These operational considerations established three design attitudes toward the XNQL translator. First, it should be modular, and thereby capable of achieving a high degree of flexibility in the sense that drastic changes can be made to one module without a need to change others. Second, it should be expandable, that is, one should be able to augment the translator to handle new target DBMSs without affecting the existing modules. Third, it should confine its target dependency to as few modules as possible ("target-specificity hiding") [Parnas 72].

An important side-effect of the modular, family approach has been the insight it provides for the design of data manipulation and structuring facilities for DBMSs. It is impossible to design a simple, coherent translator family without abstracting the essential properties from various target systems and recognizing their commonalities as well as their differences.

Based upon the previous discussion, we conclude that our major task in building the query translator for an NVDM is design of a good general framework, i.e., a consistent, efficiently implementable translator allowing effective use of the target DBMS facilities.

With these system level considerations out of the way, we proceed in the next subsection with a description of the construction of a translator aimed at achieving the above goals.

4.6 <u>XNQL Translator Specifics</u>

So far we have been occupied with basic questions of a functional nature and the general design goals of NVDM query translation. Let us now consider a single translator in some depth, namely, that for XNQL.

4.6.1 Overview

The basic approach taken by the XNQL translator is to accomplish the complex semantic manipulations required for the translation by means of step-by-step transformation of an appropriately chosen internal representation of the input text. We have chosen a tree as the intermediate representation because of the requirement for flexibility in handling a wide range of target DMLs and data structures.

Each transformation takes us somewhat closer to the target query by either: i) changing the original form of the input text to uncover the underlying "basic structure" of the query tree which characterizes the system-independent organization of the query, or ii) reshaping the basic tree to incorporate the surface structure of the target language. This transformational approach reduces overall translator complexity and the resulting modularization of the translator allows a simple, consistent design [DeRemer 76].

The translation process is (vertically) segmented into five phases as illustrated in figure 13.

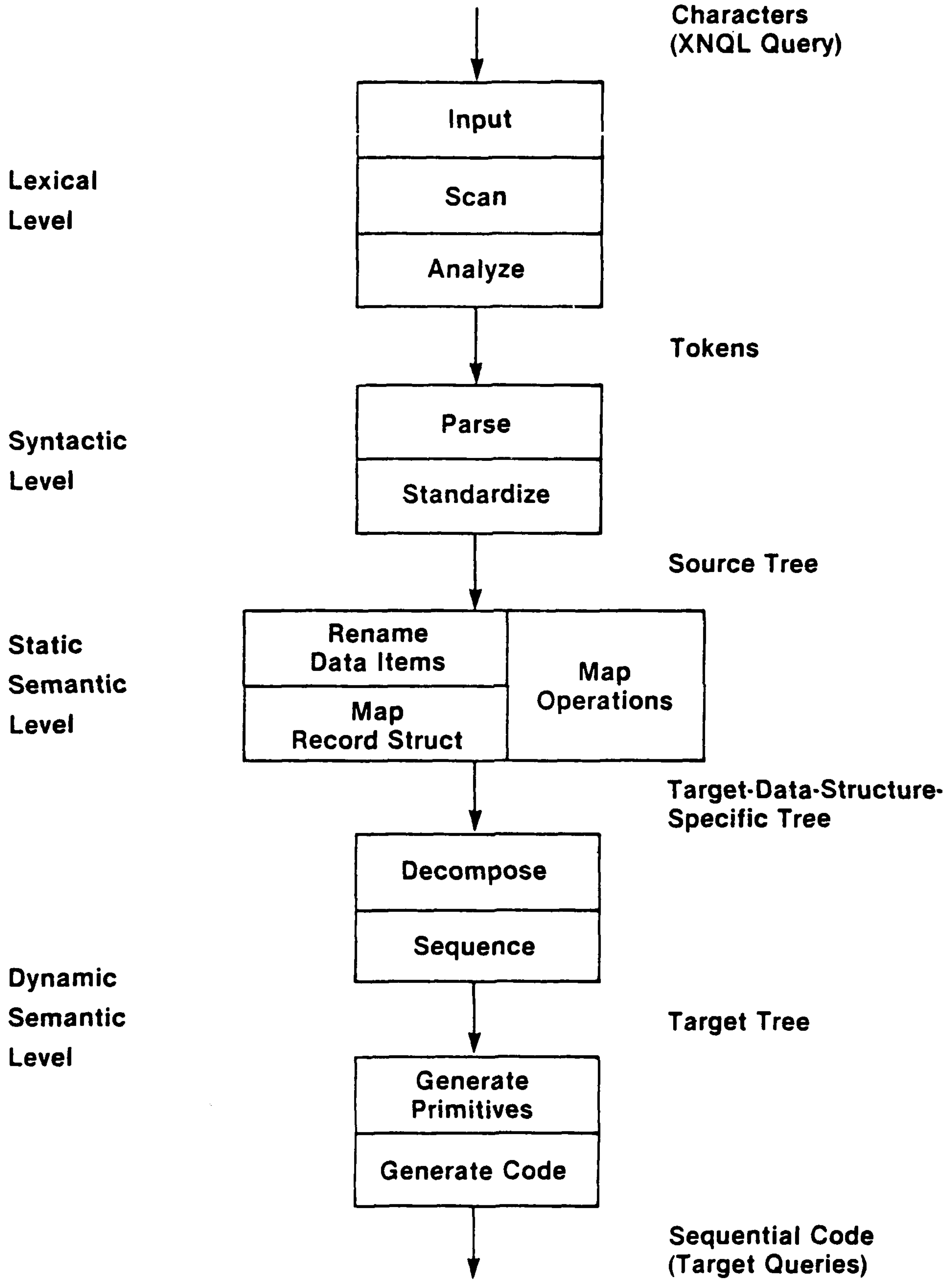


FIGURE 13. THE XNQL TRANSLATOR AS A TREE TRANSFORMER

Each box in the above diagram represents a major processing step. The vertical arrows represent the data dependencies between them. These boxes are further refined into sub-boxes with vertical and horizontal lines. Horizontal separations indicate the order in which the sub-boxes must be executed; vertical separations indicate the absence of such constraints.

For example, the "lexical level" box is divided into three components that operate serially on user inputs: one that reads the character string, one that scans the characters and one that recognizes the reserved words. An explanation of the general attributes of these five phases of the translator follows.

4.6.2 Lexical and Syntactic Analysis

The tasks of the lexical and syntactic analysis modules are conventional [Gries 71]. They produce a source(XNQL)-specific syntax tree representation of the input query. This tree contains all the information originally present in the source text as well as all the information that is inherent in the XNQL grammatical description. The source syntax tree is the first of a sequence of trees used in the translator as intermodular data structures. Each later module takes as input the tree produced by the previous module and leaves a tree that is closer to the target query by reshaping the tree, pruning source-specific information from the tree and/or incorporating target-specific information into the tree. The basic task facing the translator writer is disentangling those aspects of the source and target queries that reflect "essential" (language-independent) logical structures from those that characterize "incidental" (language-specific) representational details.

4.6.3 Standardization

Processing beyond the syntactic level can be made simpler if the source syntax tree is transformed into a standard form where each WHERE clause is represented as a binary tree of predicates connected by AND and OR nodes arranged in conjunctive normal form [Stonebraker 76].

4.6.4 Static Semantic Processing

Each XNQL query interacts with a data space which is the Cartesian product of several relations subject to the restriction of the "where" clause. Since these restrictions are frequently such that the Cartesian product becomes an equi-join (merging of two relations based on a common column), differences in source and target structures at the record level imply different join conditions in the queries.

The static semantic level of the translator does the processing needed to account for the first two levels of data structure differences (i.e., at and below the record level - see section 4.2.1) by first resolving data item name differences and then the differences in the JOINs.

The "data item renaming" module traverses the source syntax tree from the top down, replacing all leaf references to source (i.e., NVS) data items with corresponding references to target data items and depositing their attribute information at these nodes. The "record structure mapping" module then deletes all predicate nodes representing JOINs between different source relations and inserts the appropriate JOIN predicates for target records.

A third static semantic processing module, the "operator mapping" routine, converts target-system dependent operators into explicit system-specific forms to account for the kind of (primitive DML) operator differences discussed in section 4.4.2. For example, arithmetic and comparison operations on floating point numbers are transformed into operations in significant digit mode and interval mode, respectively [Yohe 79].

4.6.5 Dynamic Semantic Processing

The transformations which occur at this level account for the differences in the logical structures of the source and target query languages. Since the unit of data structure for each target query may be smaller then for XNQL (e.g., each Codasyl DML statement can only involve a single record (or set) type, whereas there is no limitation to the number of different tuple types (relations) an XNQL statement can manipulate), we first decompose the query tree into sub-trees, each of which involves a single unit of data structure that a target query can handle. The "sequence" module then chains the sub-trees together in the order that the corresponding queries should be sequenced for the target DBMS and selects the execution sequence of these chains that minimizes the number of intermediate records required for processing.

4.6.6 Code Generation

This is the final phase of the translator. Output consists of the desired target DML statements that can be executed by the local DBMSs. The first module interprets each of the sub-trees along the chains produced by the "sequence" module (see section 4.6.5) and generates CALL statements to primitive target database operations. The second (code generation) module then expands these CALLs into sequences of actual target DML statements.

The exact form of the primitives depends upon the particular target system we are considering. Their behavior characteristics fall, in general, into the following categories: search or return the first/next instance of a specified record-type, test the truth value of some predicate expression of the record-type, partition all instances of a record-type on the basis of some data item values, and evaluate aggregate functions for the specified record-type. (These correspond roughly to the information algebra [Codasyl 62] operations of searching/returning the first/next point of a line, bundling, glumping and evaluating functions of lines.)

This extra level of indirection before the actual code generation allows us to separate out the representational details of the target DMLs and makes it possible to have a standard set of primitives for each general class of target systems, that is, Codasyl, relational calculus and relational algebra systems.

The decision to set the primitives at a fairly procedural level (namely, one record instance at a time) is due largely to our desire for flexibility and power in expressing a variety of access strategies. This allows easy incorporation of optimization modules which select the "best" access paths for the input query based upon knowledge of how the records are stored (keys, inversion, indices) [Lorie 79]. This is particularly important, since the value and usefulness of XNDM in a real environment depends critically upon its performance. Experience with current relational DBMSs indicates that some form of optimization is essential in bringing performance to an acceptable level [Smith 75].

4.7 Status and Iterim Conclusions

Work on the XNQL translator is still in progress. The current version handles the two XNQL constructs which constitute the most primitive "adequate" set for database accessing (selection of columns and selection of rows based on 1- and 2-table predicates). The target systems are: i) the Multics Relational Data Store (MRDS) [Honeywell 78], a relational calculus system, and ii) the Honeywell 600/6000 Integrated Data Store (IDS) [Honeywell 71], a Codasyl-like system. These systems were chosen because they represent, as far as NVDM implementation is concerned, the two extreme models of DBMS design. For MRDS, the translator can handle all target data structures in a very general way, but for IDS, target records with multiple owners and multiple members are excluded.

The current state of XNDM is not only unfinished but also limited in scope. Many aspects of NVDMs are left untouched. Many others have only received cursory attention, or have not been explored in sufficient detail to provide a proper understanding of the problem.

For example, how should partitioned and/or replicated data be handled? How can location and replication transparency be provided? When the individual databases contain information with somewhat different (real-world) meaning, how can we resolve their differences? What can we do to improve performance, to make the system more usable, more reliable and more maintainable?

The list could be extended almost indefinitely. All of these questions are interesting and very relevant, but beyond the scope of this report. We do believe it describes some of the major and so far unexplored issues in the design and implementation of network virtual data managers. Our hope is that this work will only be the beginning, and that it will stimulate other investigations into this increasingly important area [Rothnie 77], [Gray 79], [Lindsay 79].

The preceding has described an approach for supporting uniform queries across multiple remote heterogeneous DBMSs. A major remaining requirement is a means for transmitting the resulting (structured) data to the user in a meaning preserving way.

5. DATA TRANSFER PROTOCOLS (DTPs)

This section describes a basic Data Transfer Protocol required in order to support either the transmission of data generated by a DBMS in response to a query or that generated by a process in performing an update. Key issues which must be considered are the nature of the protocol itself, translation/transformation functions which are provided to ensure the preservation of meaning, relation to other work, and supporting issues.

A data transfer protocol provides the means for moving data between possibly heterogeneous systems. Implementing one requires consideration of two major issues: i) the nature of the protocol, and ii) the transformations which are provided to assure preservation of meaning.

Sunshine Sunshine 80 provides an up-to-date definition of a protocol. As discussed therein, the key features are:

1. definition of the <u>service</u> to be provided,
2. description of the <u>entities</u> (site specific components) which collectively provide these services,
3. specification of the <u>message exchange protocol</u> used in communicating between entities, and
4. identification of the <u>lower level support functions</u> which are assumed (equivalent to specification of the services provided by lower protocol levels).

A key component of the message exchange protocol part of a Data Transfer Protocol is the specification of intermediate data formats used in transmitting data between systems. Because of its complexity, this represents a significant additional dimension to the commonly required message exchange protocols. It reflects the need to consider both the representation of structured data as well as the collection of messages to be exchanged.

Context for developing a DTP comes from two major sources. The first is some of the existing literature on File Transfer Protocols (FTPs) [Forsdick 79], [ARPAN 77], [INWG 77]. Although these protocols have observed the desirability of transferring structured data, their implementations, where existent, have been directed toward transmission of text (character string) or binary files. They are, therefore, inadequate for meeting the requirements which a DTP must satisfy.

A segment of the database literature has studied the problem of translating databases in detail [Navathe 76], [SDDTTG 76], [Shu 77]. The approaches which have been developed, together with the prototype implementations, have been rather powerful in their ability to handle complex data structures and their interrelationships as commonly found in the database context. However, this power is achieved, in part, through relying on an offline approach, which is appropriate when an entire database is to be moved between systems.

Given these two separate developmental threads, the issue arises as to whether a protocol can be developed which is sufficiently powerful to meet the needs for transmitting structured data between heterogeneous systems and, simultaneously, is sufficiently straightforward to permit the online operation common to database accesses.

We believe the answer to the preceding question is yes! In support, the following discussion provides a detailed description of the basic nature of one Data Transfer Protocol. Although this discussion has a general, issues-oriented flavor, section 5.5 describes a closely related capability implemented as part of an Experimental Network Operating System. Thus, a data point showing feasibility also exists.

5.1 DTP Services

Defining DTP services requires considering three major issues:

1. protocol invocation,
2. source/destination, and
3. transformations.

Within this section we shall consider only the first two of these issues. The complexity of the third is sufficient to require the relatively detailed discussion presented in the following section.

5.1.1 Protocol Invocation

The ARPANET protocols ARPAN 78 provided a basic point of departure for much subsequent protocol related work. Unfortunately, the basic model which seems to have been employed in their development assumed a rather high level of general computer communications knowledge by a terminal user.

By assuming users to be knowledgable, the protocol implementers were permitted to substantially reduce their concern for providing a uniform environment across systems. Thus, the set of implemented commands, as well as their representation, invocation, and responses (both to note completion of normal processing as well as to signal the occurrence of an error) varied from system to system. These variations were sufficiently small to be classified as a minor annoyance by typical ARPANET users. For mission-oriented users assumed to have little, if any, general interest in computers or communications per se, such annoyance can assume major proportions. Unsubstantiated comments leave the impression that the result may be a tacit decision by users to boycott the system.

It was very reasonable to assume users to be terminal users in the early ARPANET days and is probably still reasonable for a broad majority of users. However, developing complex distributed applications requires a process invocable protocol. This, in turn, requires some changes in the protocol design and implementation to handle the underlying asynchrony of operations. It also requires uniform responses across systems.

Handling asynchronous operations proved simple for terminal users. Effectively, one only needed to know whether type ahead was permitted. Typically, ARPANET protocols do not permit type ahead. As a result, the user must enter one parameter of the invocation sequence and wait until the appropriate response has been received to enter another. The resulting need to enter a multi-parameter command using a parameter at a time style proves difficult to program around even if a terminal emulation device such as the NAM [Rosenthal 78] is used to permit program invocation of terminal oriented protocols.

The conclusion is that Data Transfer Protocols suitable for supporting remote interaction with DBMSs must be explicitly designed to be program invocable.

5.1.2 Source/Destination

For File Transport Protocols the source and destination were files within the file systems. This was a reasonable, if not the only choice for terminal users. Database translation, in contrast, assumed an entire database which was usually spread across a large number of files at both the source and destination. Program invocation of a DTP results in a third major source/destination type: buffers within primary memory.

We exclude databases as source/destination candidates for a DTP. Files and buffers remain. Before the rapid price declines in the cost of primary memory (one megabyte of primary memory for the IBM 4300 now costs approximately $15,000), there was an obvious distinction between files and buffers. Files were assumed to be large, while buffers were small. Thus, it was reasonable to think of two separate protocols; one supporting file transfer, the other supporting buffer transfer.

Distinguishing between files and buffers based on size becomes meaningless in an environment in which large systems may have primary memory sizes of 10-100 megabytes while small systems may only have 64K words of primary memory and a disk storage capacity of less than 10 megabytes.

We conclude that only one protocol should be used for transferring files and buffers. The only difference occurs in specifying the explicit source or destination. From an implementation viewpoint this approach is very natural, since implementing a protocol for transmitting structured files requires the ability to transmit structured data within a buffer.

Given the preceding, the following discussion of a Data Transfer Protocol assumes that either source or destination may be a file or buffer. The complexity of the data interrelationships which are to be supported must now be considered.

5.2 Data and Its Translation/Transformation

This section addresses three major issues: i) the logical complexity of data whose transmission is to be supported by a DTP, ii) existing approaches to translating and transforming such data, and iii) a specific approach based on using a Common Network Representation for data being transmitted between systems.

Data being transmitted between systems can be divided into three levels of increasing complexity of which only the first two are reasonable candidates for DTP support. These are:

1. Unstructured data corresponding to transmission of some number of instances of a given data element type, e.g. text files or binary strings.

2. Structured data without pointers corresponding to transmission of a collection of structured but independent records having the same logical structure (tree representation and data element type), and

3. Structured data with pointers between records.

We believe that a DTP should support transfer of the first two categories of data. Although pointers are, conceptually, just another data type, significant care is required in their interpretation. This reflects the application dependent nature of their specification and interpretation. Thus, the decision to support transfer of data in the third category may be application dependent.

Translating the first two categories of data requires considering two separate but interrelated issues. The first is the complexity of the transformations which may be performed on the data. The second is the extent to which the semantics of system data representations are "known" by the transformer and, thereby, do not have to be explicitly stated by the user.

We assume that data exists at a Sender (either file or buffer) and is to be transmitted to a Receiver (also either file or buffer). Additionally, for symmetry and conceptual simplicity, we assume that the invoker of the transfer may be co-located with the Sender, the Receiver, or may exist on a third system distinct from either Sender or Receiver. Moreover, we assume that the Receiver data may be only a logical subset of the Sender data. That is, each Receiver data element value can be a functional image of one or more Sender data element values. This level of capability proves sufficient to handle the transformations required by XNOS. Moreover, it is sufficiently restrictive to permit the protocol to assume responsibility for data element type mappings and for mapping data structures to their linear representations (cf. the discussion of XNOS in section 5.5).

A general purpose programming language provides the most general means for performing source to destination data maps. The disadvantages of this approach are equally obvious, since each different transformation requires writing a different program.

The ARPANET Data Reconfiguration Service (DRS) [Cerf 74] provided a transformation capability which was nearly as general as that provided by a general purpose programming language. Additionally, specification of the transformations was simplified by providing a compact and concise notation. However, before a user could invoke transformations, a programmer was responsible for both explicitly stating the data type representations used by source and destination systems as well as for procedurally specifying the means for establishing the correspondences between the logical structure of the data, say a tree (commonly used to represent records), and the serial representation of the corresponding data elements on a storage medium.

DSCL [Schneider 75] provided a somewhat simpler means for specifying data transformations. However, the user was still required to explicitly indicate the transformations to be performed on the bit string representation of the data.

Historically, database translation has provided extensive incorporation of data type and structure semantics. However, the complexity of the requirements for supporting database translation substantially exceed those of the transformation portion of a data transfer protocol. Nevertheless, the relevant database translation literature, e.g., the University of Michigan Data Translator [Navathe 76], the Stored Data Definition Translation Task Group report [SDDTTG 76], and the Define/Convert capabilities [Shu 77] provides important insights into the levels of sophistication which can be achieved.

Instead of specifying data transformations procedurally, one would prefer to specify the source and destination record descriptions together with their interrelationship and require that the system establish the appropriate transformations. Within the context of supporting network access to data, this objective is realizable as described in 5.5 in the context of XNOS. Thus, the user need only specify the relevant data structures using the specification technique of the programming language being used.

5.3 Implementing a Data Transformer

Data translation is required both to preserve the logical structure of data being transmitted between heterogeneous systems and to preserve the type and value of individual data elements. Structure preservation is required, since different systems may store the same logical record in different ways.

The structure of the data translation process is determined by: i) the site(s) at which translation is performed, and ii) the decision as to whether to use a Common Network Representation.

5.3.1 Translation Implementation Alternatives

Three major translation implementation alternatives exist: i) all translation is performed at the source, i.e., the source translates data directly to the system dependent representation used by the destination, ii) all translation is performed at the destination, and iii) translation is distributed between source and destination.

If either of the first two alternatives are employed, one can use direct source-destination translation. The cost, however, if there are N heterogeneous systems, is that of building N*(N - 1) translators. To reduce this cost, one may opt instead for using a Common Network Representation (CNR) and distributing the translation between sender and receiver.

Using a CNR results in translating the source data to a common representation for transmission through the network. It is is then retranslated to the form appropriate for the destination. Apparently, this approach only requires the construction of 2N translators (one to and one from each of the N different system types). If the host dependent portion of the translation process can be represented via tables, only two basic translators are required together with the appropriate code. (This is the approach used in the discussion of XRRA described in section 5.5.)

The disadvantage of this approach, in comparison with direct sender-receiver translation, is the possibility of reduced performance. In weighing the advantage of implementation simplicity against this possible disadvantage, two key issues are the expected bandwidth of the transmission process and the likelihood that the supported systems will change. Since the bandwidth is likely to be limited when terrestrial communication lines are employed, the overhead of using a Common Network Representation may be negligible. Moreover, if the characteristics or requirements of the supported systems is likely to change, the simplification resulting from the CNR approach may prove very desirable.

5.3.2 Translation as a Distributed Process

Describing the distributed translation process is facilitated by introducing some notation.

1. SSR is the Sender Storage Representation of the record and corresponds to its serial representation in the Sender-Buffer.

2. SES is the Sender Element Sequence of the record defining the sequence in which data elements are encountered as one traverses the SSR.

3. SCNR is the Sender Common Network Representation of the record; the element sequence of the SCNR is that of the SES but the individual data elements have been translated into their common network-wide representation.

4. RES is the Receiver Element Sequence of the record.

5. RCNR is the Receiver Common Network Representation of the record.

6. RSR is the Receiver Storage Representation of the record.

Given the preceding, the translation process can be simply described as a function t:SSR ---▶ RSR. Provided a common intermediate representation is to be used, t can be viewed as the composition of three functions, s, m and r, where s is the Sender translation function, m is the Sender-Receiver transformation function, and r is the Receiver translation function. That is, $t = r \circ m \circ s$, where $\circ$ denotes the composition function.

Sender Translation

The Sender translation function $s: SSR \dashrightarrow SCNR$. The input to the translation process consists of three components: i) SSR, ii) a Sender Logical Record Description (SLRD) describing the SES, and iii) a Sender Representation Table (SRT) describing the representation of individual data element types at the Sender. The output consists of the SCSR in Sender element sequence.

Implementation of the translation process is conceptually straightforward and simply consists of: i) reading the SLRD to determine the type of the next data element, ii) extracting the appropriate number of bits beginning from the current position of the translation pointer, with offset determined by the data element type, iii) performing the appropriate translation and concatenating the result to the existing interim SCNR, iv) advancing the translation pointer appropriately, and continuing the process until the SSR has been processed.

Sender-Receiver Transformation

Sender-Receiver transformation should permit: i) different Sender and Receiver names for the same component, ii) deletion of Sender data elements (because the requestor may not have the appropriate security authorization or, alternatively, may not require a given component), iii) specification of a Receiver data element as a function of Sender data elements (cf. the discussion in the following section), as well as iv) modification of the type of a data element (for example, it may prove desirable to convert an integer data type to a real data type).

The Sender-Receiver transformation $m:SCNR \longrightarrow RCNR$. The four inputs to the transformation process are: i) SCSR, ii) Sender Logical Record Description, iii) Receiver Logical Record Description, and iv) a Name Map Table.

The need for (i)-(iii) is evident. The Name Map Table is required to permit Sender and Receiver to use different names for the same data element and to specify the appropriate map between their data elements. Its existence adds substantial flexibility to the transformation process and, in general, supports transformation whenever the RCNR is a function of the SCNR.

The preceding discussion has not addressed the issue of where transformation is performed. Either Sender or Receiver is appropriate. Implicitly, we are assuming transformation to SCNR is performed at the source while transformation from RCNR is performed at the destination.

The SCNR --. RCNR map may be performed in either place. This approach should be contrasted with the approach used in a Network Operating System supported environment as described in section 5.5.

Receiver Translation

The Receiver translator r:RCNR ---► RSR. The inputs to this translation process are: i) RCNR, ii) Receiver Logical Record Description, and iii) Receiver Representation Table. Logically, this translation process is equivalent to the inverse of the source translation.

5.4 A Data Transformation Protocol

This section describes a basic Data Transfer Protocol meeting the objectives and constraints established in the preceding. The basic service to be provided is preservation of meaning in transmitting structured data between heterogeneous systems. The entities required to provide this service are discussed at length below as is a transaction flavored message exchange protocol. Implementation of this protocol assumes a means for error free reliable transmission of data between systems. Protocols providing such functions are sometimes termed host-host protocols; the basic requirements correspond to the services provided by levels four and five of the ISO Reference Model [ISO 79].

We begin by describing the DTP environment and continue with a discussion of the operation of the protocol viewed as a distributed collection of processes. Thereafter, the specifics of the message exchange process are discussed.

5.4.1 DTP Processing Sequence

We assume that some process, perhaps a Network Virtual Data Manager, has generated one or more records to be transmitted to another computer system. Destination records are to be images of Sender records.

The DTP design which we will describe uses a major simplifying assumption: the elimination of options negotiation. This is achieved by requiring that all parameters be transmitted together with the record. Thus, the protocol initialization process which is usually required to ensure that Sender and Receiver are operating with compatible parameters is avoided.

DTP operation requires five separate processing phases.

- o Translation of Sender Record
- o Sender$\rightarrow$Receiver Transmission
- o Translation of Receiver Record

- Receiver Process Notification
- Acknowledgment to Sender

Sender and Receiver translation are logically symmetric invocations of the data transformation process discussed in 5.3. Sender$\rightarrow$Receiver transmission is performed in response to an invocation of the message exchange protocol discussed below.

Process notification is required to guarantee that the appropriate process within the Receiver is notified upon completion of transmission of the record(s). If the process is in the WAIT state, this could be achieved through forcing its rescheduling for the processor. If the process is ACTIVE, a flag can be set or a message sent to an appropriate port of the process.

The acknowlegement phase is realized via transmission of a message from Receiver to Sender. Its receipt by the Sender component of the protocol is required to permit appropriate housekeeping operations, e.g., flushing of buffers before transmitting additional record(s).

5.4.2 DTP Commands

Our discussion of a DTP assumes: i) protocol invocation by a process within the sender, ii) the protocol is passed either a file name or a buffer name plus access path specifications for both Sender and Receiver which will be referred to below as Sender-Name and Receiver-Name, and iii) the protocol is passed a pointer to a parameter list (discussed below). If we assume flow control is performed by a lower level protocol, only four commands are required.

<u>Sender--►Receiver</u>

```
SEND(Id,Sender-Location,Receiver-Location;
     Param-List;Return-Code)

ABORT(Id)
```

<u>Receiver--►Sender</u>

```
RECEIVED (Id)

REJECT (Id)
```

SEND is the basic command for transmitting data from Sender to Receiver. The Return-Code notifies the requesting process whether the transmission was successfully completed. ABORT is required to permit premature termination when required. RECEIVED indicates that transmission was successfully completed. REJECT indicates unwillingness of the Receiver to accept (or support completion of) transmission.

The Id provides a compact mechanism for detecting duplicates and referring to a specific request. A variety of techniques exist for selecting such identifiers; one is the use of a system time stamp having sufficient resolution.

The parameter list for the DTP consists of two major components supporting record translation and transformation. The basic information requirements for their support were discussed above.

It is reasonable to ask if four commands are sufficient to support a reasonable Data Transfer Protocol. A definitive answer clearly requires user experience. However, existing knowledge indicates that the total number of required commands can be kept relatively small. Two factors support this assertion. The first is elimination of options negotiation resulting from the use of a parameter list. The second is the observation that although contemporary FTPs have a large number of commands, this reflects their support of three major classes of functions: i) data transfer, ii) file manipulation, and iii) remote job execution INWG 77 . The number of commands supporting the data transfer portion of the FTP is thus substantially smaller than the number of FTP commands.

Heafner and Nielsen [Heafner 80] highlight the wide variance in commands supported by FTP protocols. In examining five different FTP specifications, a total of 170 distinct features (corresponding roughly to commands or distinguishable parts of multifunction commands) were identified. Only eight features were common to all five FTPs.

5.5 An Example--the NBS Experimental Network Operating System

The preceding has established the basic mechanisms required for exchanging structured data. Acceptance of the issues and approaches which were discussed requires assurance that they are, indeed, implementable. This section describes an Experimental Network Operating System (XNOS) implemented at the National Bureau of Standards (NBS) and providing, as a component, such a capability.

5.5.1 XNOS Overview

XNOS has been described in detail by Kimbleton [Kimbleton 78]. Although the implementation approach described therein has been discarded to achieve compactness and efficiency, the basic collection of primitives is unchanged. The following brief discussion may be viewed as a summary.

XNOS functionality can be divided into two basic categories: i) provision of a uniform user viewpoint across heterogeneous systems, and ii) provision of a uniform mechanism to support intersystem interaction. Critical discussion of a network operating system requires prior establishment of the environment in which it is to be implemented. XNOS implementation assumes:

> Prior existence of resources - precluding implementation of a new, distributed operating system and requiring, instead, use of the existing operating systems on those systems supported by the network operating system,
>
> Incision minimization - modifications to operating systems are strongly discouraged to encourage operational acceptance of network operating systems in a mission oriented environment,
>
> Offloading - to minimize local host overhead in supporting a network operating system, and
>
> Uniform network wide resource access and control language - requiring network users to learn the XNOS command language rather than supporting direct translation between the local and remote system command languages for resource access and control.

The initial XNOS prototype implementation supported the Multics, TENEX, TOPS-10, and UNIX operating systems attached to the ARPANET. Moreover, offloading was accomplished through implementing essentially all software in the programming language C on a PDP-11/45 running UNIX. Figure 14 illustrates this implementation environment.

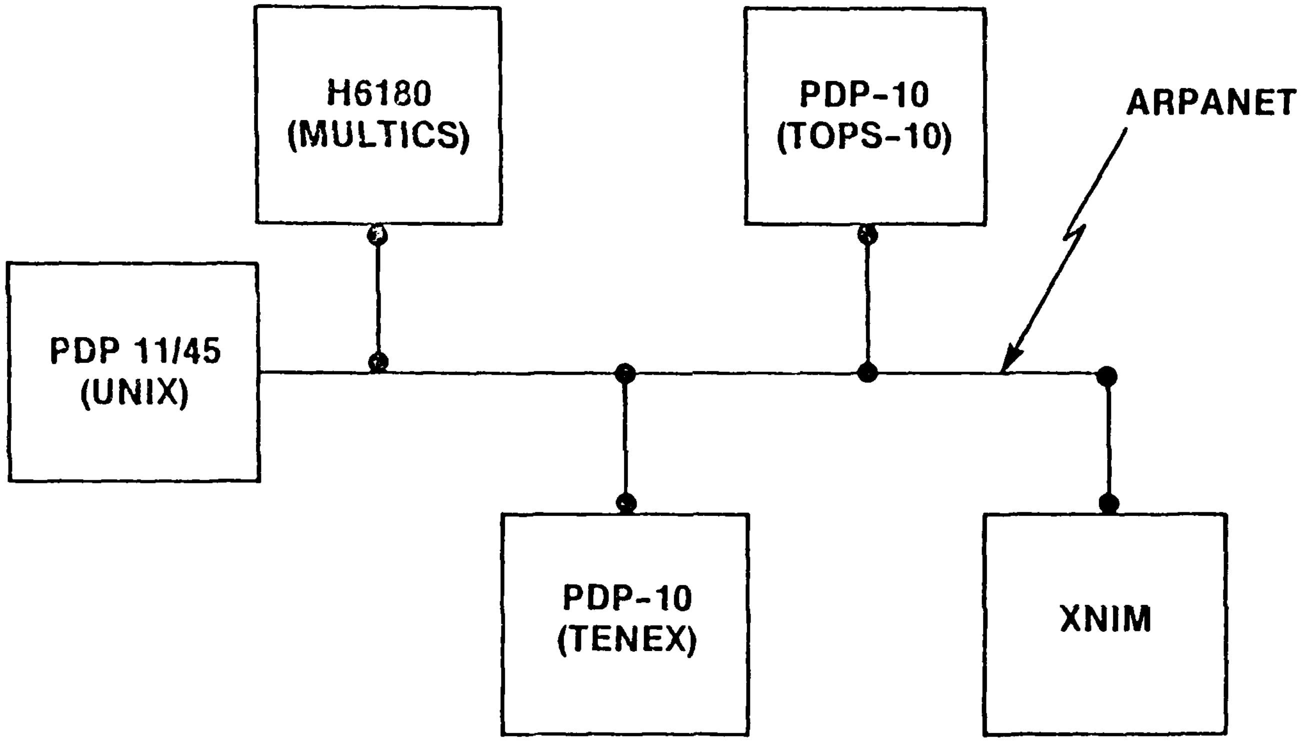


FIGURE 14. XNOS IMPLEMENTATION ENVIRONMENT

A user generally interacts with systems via a command language. Excluding utilities, a command language provides a basic mechanism for manipulating files and running jobs. Accordingly, XNOS provides a common command language across heterogeneous systems together with a distributed job execution capability.

The XNOS common command language for file manipulation is illustrated in figure 15.

Network Wide File Manipulation	
Intra-System	Inter-System
Append Compare Copy Create Delete Erase List Rename Type Undelete	Copy File A at Host A To File B at Host B

FIGURE 15. XNOS COMMON COMMAND LANGUAGE

The relative paucity of commands in comparison with the number usually found in an operating system command manual reflects the fact that most such commands relate to the assignment of data across devices. By definition (ours!) such commands are inappropriate for the network user interested in accessing resources and uninterested in the intricacies of systems configurations. Since the distributed job execution capability presumes no-setup jobs, only a single command is required. Multi-step jobs can be handled by stacking a series of individual commands. To facilitate user entry of such commands, a prompting facility is provided.

5.5.2 Supporting Uniform System-System Interactions

System-system interactions are supported within XNOS by two major functional components: i) a relatively crude network wide interprocess communication mechanism, and ii) a Remote Record Access mechanism for preserving the meaning of structured data being transmitted between heterogeneous systems.

XNOS interprocess communication only provides a mechanism for two processes located in different systems to communicate. It does not provide an ability for remote process invocation and control nor does it provide synchronization capabilities. Since the existing mechanism does not represent a contribution to advancing the current state of the art, we shall not describe it further. Functionally, the primitives which it provides represent a limited subset of those provided by MSG [NSW 76]. Implementation is via Telnet links rather than raw connections, however.

A detailed discussion of XNOS Remote Record Access (XRRA) is given by Wood [Wood 80]. Support of meaning-preserving transmission of structured data within XNOS was constrained, to some extent, by limitations in existing ARPANET protocol implementations. Thus, the following description emphasizes the logical nature of the interaction and adds appropriate comments indicating problems requiring additional consideration.

Describing XRRA is facilitated by discussing the major sequence of processing actions which must be performed in transmitting data between a system (Sender) containing a buffer of structured data and a Receiver system. A logical overivew of XRRA is contained in figure 16.

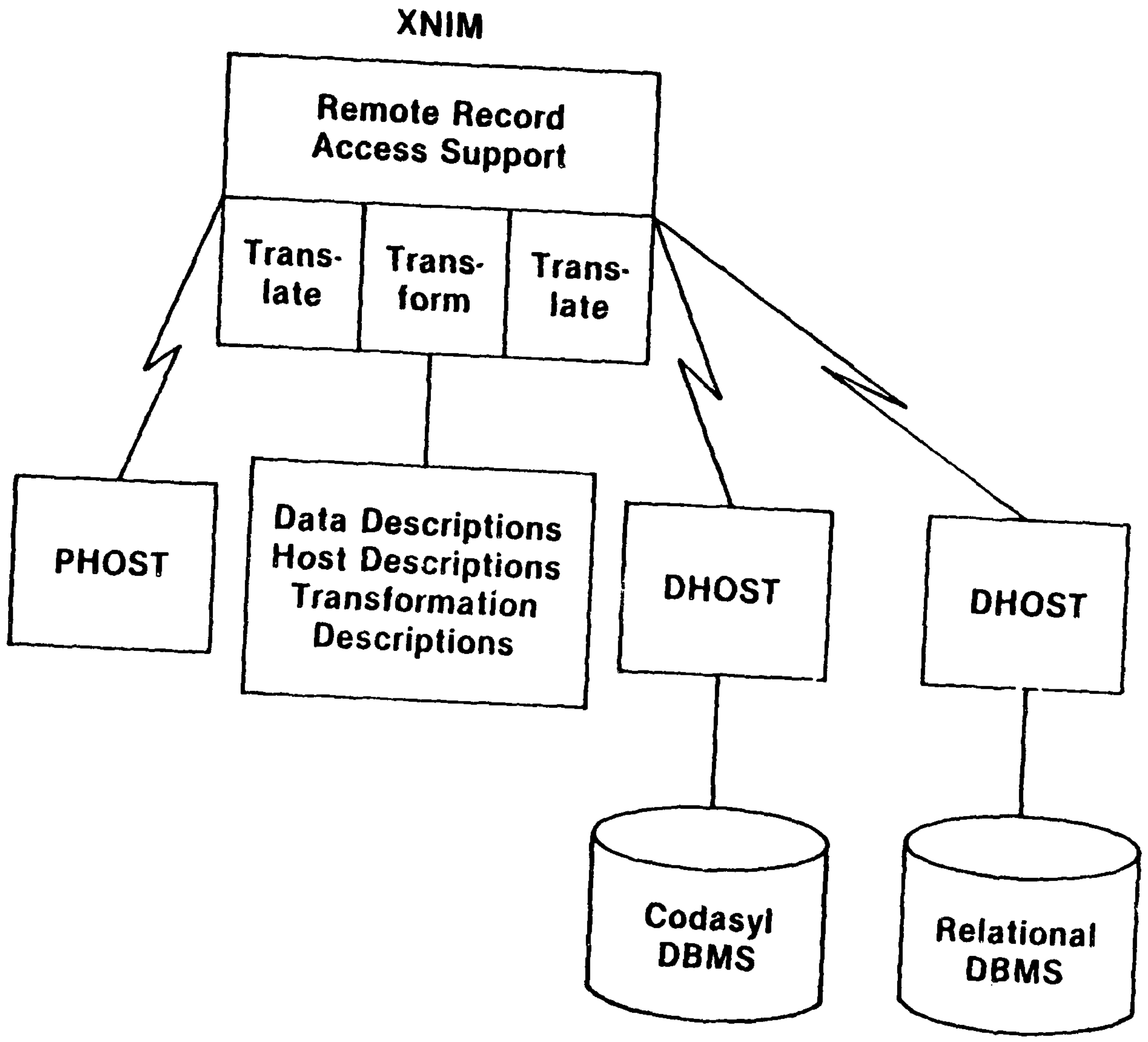

XNIM
Remote Record Access Support
Trans-late
Trans-form
Trans-late
PHOST
Data Descriptions Host Descriptions Transformation Descriptions
DHOST
DHOST
Codasyl DBMS
Relational DBMS

FIGURE 16. XRRA LOGICAL STRUCTURE

The XRRA assumes: i) the Sender data exists within a buffer, ii) a Sender Logical Record Description exists, and iii) data type encodings as well as system/language dependent information describing how the graph of the record is mapped to its buffer representation exist. Items (ii) - (iii) are assumed to be maintained by XNOS as part of XRRA on the Experimental Network Interface Machine (XNIM) indicated in figure 14.

Data generated by the sender is transmitted to the XNIM where it is converted to a Common Network Representation termed a Network Normal Form in the XNOS terminology. After conversion, a name map table is used to effect the appropriate mappings between Sender and Receiver data. Thereafter, the data is reconverted to the form appropriate to the Receiver.

Figures 17 - 19 illustrate this process. Figure 17 describes the record as it exists on the Sender system; figure 18 describes the record as it is to exist on the Receiver system; and figure 19 shows the name map table establishing the interrelationship between Sender and Receiver records.

ID	=	1.1	SNO	INT
ID	=	1.2	NAME	CHAR (15)
ID	=	1.3	STREET	CHAR (32)
ID	=	1.4	CITY	CHAR (32)
ID	=	1.5	STATUS	CHAR (8)
ID	=	1.6	PNO	LONGINT
ID	=	1.7	QUANTITY	INT
ID	=	1.8	PNAME	CHAR (8)
ID	=	1.9	COLOR	CHAR (12)
ID	=	1.10	WEIGHT	REAL
ID	=	1.11	LOCATION	CHAR (20)

FIGURE 17. SENDER LOGICAL RECORD DESCRIPTION

ID	=	1.1	SNO	INT
ID	=	1.2	SNAME	CHAR (15)
ID	=	1.3	SSTAT	CHAR (8)
ID	=	1.4	SCITY	CHAR (20)
ID	=	1.5	PNO	INT
ID	=	1.6	PNAME	CHAR (15)
ID	=	1.7	PCOL	CHAR (12)
ID	=	1.8	PWT	REAL
ID	=	1.9	PLOC	CHAR (20)
ID	=	1.10	QUANT	INT

FIGURE 18. RECEIVER LOGICAL RECORD DESCRIPTION

SNO	=	SNO
SNAME	=	NAME
SSTAT	=	STATUS
SCITY	=	CITY
PNO	=	PNO
PNAME	=	PNAME
PCOL	=	COLOR
PWT	=	2.2*WEIGHT
PLOC	=	LOCATION
QUANT	=	QUANTITY

FIGURE 19. NAME MAP TABLE INTERRELATING SENDER AND RECEIVER LOGICAL RECORD DESCRIPTIONS

XRRA representation of boolean, binary and character data is unsurprising. Representation of reals and integers is accomplished through transformation to the equivalent ASCII character string. This has the disadvantage of expanding the amount of space required for their representation. It has the advantage of assuring that no information is lost in performing arithmetic operations indicated in the name map table. This reflects the possibility of using character based, arbitrary precision arithmetic packages. Note that this does not eliminate the round off errors which are inevitable in going from say, a machine with a word length of 64 bits to one with a word length of 16 bits. However, it does assure that no precision will be lost within XNOS.

6. CONCLUDING REMARKS

The preceding has structured an integrated approach to supporting network access to data which provides both a uniform query capability across multiple remote heterogeneous DBMSs, as well as the ability to preserve the meaning of structured data being transmitted between heterogeneous systems. The feasibility of the basic approach has been established through the ongoing implementation of prototype systems which were also discussed.

The importance of NVDMs is evident from the increasing interest in the topic [Cardenas 79],[Chu 79], [Esculier 79],[Fauser 79],[Tsubaki 79]. We believe this interest reflects the requirement for rapid incorporation of new and more sophisticated database products within an operational environment.

The NVDM design and implementation requires considering two major classes of issues: distribution management, which has not been addressed in this report, and heterogeneity management, which has been the focus of the paper. Distribution management deals with issues related to supporting concurrent access to multiple DBMSs and has not yet been addressed in the context of XNDM. However, the existing literature on distributed DBMSs provides substantial insight into such problems [Bernstein 79].

Almost all current DMLs for generalized DBMSs support updates as well as queries [Date 77]. The preceding has only considered queries since this is required to establish addressability to existing data. Considering updates requires handling modification of existing data, insertion of new record instances and deletion of existing record instances. Since the network user may only see a limited portion of the data supported by a local DBMS, the problem of updating views must also be considered. Although the general solution to this problem is very difficult [Dayal 79], partial solutions sufficient for the requirements of many network users may provide an alternative.

7. REFERENCES

[ANSI/X3/SPARC 75] Study Group on Data Base Management Systems: Interim Report. FDT (Bulletin of ACM SIGMOD) Vol.7, No.2 (1975).

[Anthony 65] Anthony, R. Planning and Control Systems: A Framework for Analysis. Boston: Division of Research, Graduate School of Business Administration, Harvard University, 1965.

[ARPAN 78] ARPANET Protocol Handbook. NIC 7104, Network Information Center, SRI International, Menlo Park, Calif., 1978.

[Astrahan 76] Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. System R: Relational Approaches to Database Management. ACM Trans. Database Sys., Vol.1 (1976), 97-137.

[Bachman 69] Bachman C. W. Data Structure Diagrams. Data Base, Vol.1, No.2, 1969.

[Backus 78] Backus, J. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. Comm. ACM, Vol.21 (1978), 613-641.

[Biller 79] Biller, H. On the Equivalence of Data Base Schemas - A Semantic Approach to Data Translation. Information Sys., Vol.4 (1979), 35-47.

[Boehm 73] Boehm, B. K. High Cost of Software. In "Proc. Symp. on High Cost of Software", Naval Postgraduate School, Monterey, Calif., 1973.

[Brodie 78] Brodie, M. Specification and Verification of Data Base Semantic Integrity. Tech. Rep. CSRG-91, Univ. of Toronto, Comp. Sci. Dept., Toronto, Canada, Apr. 1978.

[Cardenas 79] Cardenas, A.F. and Pirahesh, M.H. The E-R Model in a Heterogeneous Data Base Management System. Proc. Intern. Conf. on Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, CA, 1979, pp. 648-654..

[Chamberlin 76] Chamberlin, D. D., Astrahan, M. M., Eswaran, K. P., Griffiths, P. P., Lorie, R. A., Mehl, J. W., Reisner, P. and Wade, B. W. SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control. IBM J. Res. & Develop., Vol.20 (1976), 560-575.

[Cerf 72] Cerf, V. G., Harslem, E. F., Heafner, J. F., Metcalfe, R. M. and White, J. E. An Experimental Service for Adaptable Data Reconfiguration. IEEE Trans. Comm. Com-20 (1972), 557-564.

[Chen 76] Chen P. P.-S. The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. Database Sys., Vol.1 (1976), 9-36.

[Chu 79] Chu, W.W. and To, V.T. A Hierarchical Conceptual Model for Data Translation in a Heterogeneous Database System. Proc. Intern. Conf. on Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, CA, 1979, p. 647.

[Codasyl 62] Codasyl Development Committee, An Information Algebra. Comm. ACM, Vol.5 (1962), 190-204.

[Codasyl 73] Codasyl Data Description Language Committee, DDL Journal of Development (1973).

[Codasyl 76] Codasyl Systems Committee, Selection and Acquisition of Data Base Management Systems (1976).

[Codd 70] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. Comm. ACM, Vol.13 (1970), 377-387.

[Codd 71] Codd, E. F. Relational Completeness of Data Base Languages. In "Data Base Systems", R. Rustin, Eds., Prentice-Hall, Englewood Cliffs, New Jersey, 1971, pp.65-98.

[Collmeyer 72] Collmeyer, A. J. Implications of Data Independence on the Architecture of Database Management Systems. ACM SIGFIDET Workshop on Data Description, Access and Control, Denver, Colorado, 1972, pp.307 - 321.

[Date 77] Date, C. J. "An Introduction to Database Systems", 2nd Ed., Addison-Wesley, Reading, Mass., 1977.

[Dayal 78] Dayal, U. and Bernstein, P.A. On the Updatability of Relational Views. Proc. 4th Intern. Conf. Very Large Data Bases, West-Berlin, Germany, 1978, pp. 368-378.

[Denning 76] Denning, D. E. A lattice model of secure information flow. Comm. ACM, Vol.19 (1976), 236-243.

[DeRemer 76] DeRemer, F. L. Transformational Grammars. In "Compiler Construction - An Advanced Course", L. F. Bauer and J. Eickel, Eds., Springer-Verlag, New York, 1976, pp.121-145.

[Esculier 79] Esculier, C. and Glorieux, A.M. The Sirius-Delta Distributed DBMS. Proc. Intern. Conf. on Entity-Relationship Approach to Systems Analysis and

Design, Los Angeles, CA, 1979, pp. 616-624.

[Fauser 79] Fauser, U. and Neuhold, E.J. Transaction Processing in the Distributed DBMS-POREL. Proc. Fourth Berkeley Conf. on Distributed Data Management and Computer Networks, Berkeley, CA, 1979, pp. 353-375.

[Forsdick 79] Forsdick, H. and McKenzie, A. FTP Functional Specification. BBN Rep. 4051, Cambridge, Mass., 1979.

[Gray 78] Gray, J. N. Notes on Database Operating Systems. In "Operating Systems - An Advanced Course", G. Goos and J. Hartmanis, Eds., Springer-Verlag, New York, 1978, pp. 393-481.

[Gray 79] Gray, J. N. A Discussion of Distributed Systems. IBM Res. Rep. IBM Res. Lab., San Jose, Calif., 1979.

[Gries 71] Gries, D. "Compiler Construction for Digital Computers", Wiley, New York, 1971.

[Griffiths 76] Griffiths, P. P. and Wade, B. W. An Authorization Mechanism for a Relational Database. ACM Trans. Database Sys., Vol.1 (1976), 242-255.

[Hammer 79] Hammer, M. and McLeod, D. On Database Management System Architecture. Univ. Southern Calif. Comp. Sci. Dept. TR No. 4, Los Angeles, Calif., Apr. 1979.

[Heafner 80] Heafner, J.F. and Nielsen, F.H. A Linear Programming Model for Optimal Computer Network Protocol Design. Proceedings of the NCC, 1980.

[Honeywell 71] Honeywell Information Systems "Integrated Data Store", Order No. Br69, Rev.1, Dec. 1971.

[Honeywell 77] Honeywell Information Systems "Multics Integrated Data Store Reference Manual", Draft, Oct. 1977.

[Honeywell 78] Honeywell Information Systems "Multics Relational Data Store (MRDS) Reference Manual", Order No. AW53, Rev.2, Oct. 1978.

[Horowitz 76] Horowitz, E. and Sahni, S. "Fundamentals of Data Structures", Computer Science Press, Woodland Hills, Calif., 1976, pp.293 - 295.

[IBM 76] IBM Corp. "IMS/VS Version 1. General Information Manual", IBM Program No.5740-XX2, Apr. 1976.

[INWG 77] A Network Independent File Transfer Protocol. Prepared by the High Level Protocol Group, IFIP, International Network Working Group, HLP/CP 1, December, 1977.

[ISO 79] Reference Model of Open Systems Interconnection. ISO/TC97/SC16 N227 Aug. 1979.

[Kent 79] Kent, W. Limitations of Record-Based Information Models. ACM Trans. Database Sys., Vol.4 (1979), 107-131.

[Kimbleton 78] Kimbleton, S.R., Wood, H.M., and Fitzgerald, M.L. Network Operating Systems--An Implementation Approach. Proc. NCC, Vol.47 (1978), 773-782.

[Kimbleton 79] Kimbleton, S. R., Wang, P. S.-C. and Fong, E. XNDM: An Experimental Network Data Manager. Proc. Fourth Berkeley Conf. on Dist. Data Mgmt. and Comp. Net., Aug. 1979, pp.3-17.

[Lavallee 72] Lavallee, P., Ohayon, S. and Sauvain R. Non-Procedural Access to DMS Databases. In "Proc. 19th Intern. XDS User's Meeting", Dec. 1972.

[Lindsay 79] Lindsay, B. G., Selinger, P. G., Galtieri, C., Gray, J. N., Lorie, R. A., Price, T. G., Putzolu, F., Traiger, I. L. and Wade. B. W. Notes on Distributed Databases. IBM Res. Rep. RJ2571, IBM Res. Lab., San Jose, Calif., Jul. 1979.

[Lorie 79] Lorie, R. A. and Nilsson, J. F. An Access Path Specification Language for a Relational Data Base System. IBM J. of Res. & Develop., Vol.23 (1979), 286-298.

[McGee 74] McGee, W. C. A Contribution to the Theory of Data Equivalence. In "Data Base Management", J. W. Klimbie and K. L. Koffeman, Eds., North-Holland, Amsterdam, 1974, p.123.

[McLeod 79] McLeod, D. and King, R. Applying A Semantic Database Model. Univ. Southern Calif. Comp. Sci. Dept. TR No. 5, Los Angelos, Calif., Aug. 1979.

[Mealy 67] Mealy, G. H. Another Look at Data, Proc. FJCC, Vol.36 (1967), 525-534.

[MIT 77] MIT Information Processing Services "The Relational Data Management System Reference Guide", 1st Ed., Jan. 1977.

[Navathe 76] Navathe, S. B. and Fry, J. P. Restructuring for Large Databases: Three Levels of Abstraction. ACM Trans. Database Sys., Vol.1 (1976), 138-158.

[Neely 77] Neely, P. M. Implementation Independent Arithmetic: Speculation for Discussion. Software - Practice and Experience, Vol.7 (1977), 461-168.

[NSW 76] National Software Works Semi-Annual Technical Report. CADD-7603-0411, Mass. Comp. Assoc., Wakefield, Mass., 1976.

[Paolini 77] Paolini, P. and Pelagatti G. Formal Definition of Mappings in a Database. Proc. ACM-SIGMOD Intern. Conf. Mngt. Data, ACM, New York, ACM, New York, Aug. 1977, pp.40-46.

[Parnas 72] Parnas, D. L. On the Criteria To Be Used in Decomposing Systems into Modules. Comm. ACM, Vol.15 (1972), 1053-1058.

[Ritchie 74] Ritchie, D. M. and Thompson, K. The UNIX Time-Sharing System. Comm. ACM, Vol.17 (1974), 365-375.

[Rosenthal 78] Rosenthal, R. and Lucas, B. D. The Design and Implementation of the National Bureau of Standards' Network Access Machine. NBS SP 500-35, National Bureau of Standards, Washington, D.C. 1978.

[Rothnie 77] Rothnie, J. B. and Goodman N. A Survey of Research and Development in Distributed Database Management. Proc. 3rd Intern. Conf. Very Large Data Bases, Tokyo, Japan, 1977, pp.48 - 62.

[Schneider 75] Schneider, G. M. DSCL-A Data Specification and Conversion Language for Networks. Proc. ACM SIGMOD, May 1975, pp. 139 - 148.

[SDDTTG 76] Stored-Data Description and Data Translation: A Model and Language. Report of the Stored-Data Definition and Translation Task Group of the Codasyl Systems Committee (1976).

[Shoshani 78] Shoshani, A. CABLE: A Language Based on the Entity-Relationship Model. Univ. Calif. Lawrence Berkeley Lab. Rep. UCID-8005, Berkeley, Calif., Jan. 1978.

[Shu 77] Shu, N. C., Housel, B. C., Taylor, R. W., Ghosh, S. P. and Lum, V. Y. EXPRESS:A Data EXtraction, Processing and REStructuring System. ACM Trans. Database Sys., Vol.2 (1977), 134-174.

[Smith 75] Smith, J. M. and Chang, P. Y.-T. Optimizing the Performance of a Relational Algebra Database Interface. Comm. ACM, Vol.18 (1975), 568-579.

[Stonebraker 75] Stonebraker, M. and Held G. Networks, Hierarchies, and Relations in Data Base Management. Mem. No. ERL-M504, Elec. Res. Lab., Coll. of Eng., Univ. Calif., Berkeley, Calif., Mar. 1975.

[Stonebraker 76] Stonebraker, M., Wong, E., Kreps, P. and Held, G. The Design and Implementation of INGRES. ACM

Trans. Database Sys., Vol.1 (1976), 189-222.

[Sunshine 80] Sunshine, C. Formal Methods for Communication Protocol Specification and Verification. NBS SP- (in preparation). National Bureau of Standards, Washington, D.C. 1980.

[Tsubaki 79] Tsubaki, M. and Hotaka, R. Distributed Multi-Database Environment with a Supervisory Data Dictionary Database. Proc. Intern. Conf. on Entity-Relationship Approach to Systems Analysis and Design, Los Angeles, CA, 1979, pp. 625-646.

[Wang 80] Wang, P. S. C. and Kimbleton, S. R. The Design of a Database Access Protocol.

[Weber 78] Weber, H. A. Software Engineering View of Data Base Systems. Proc. 4th Intern. Conf. Very Large Data Bases, West-Berlin, Germany, 1978, pp.36 - 51.

[Wood 80] Wood, H. M. and Kimbleton, S. R. Remote Record Access: Requirements, Implementation and Analysis. NBS SP 500- (in preparation). National Bureau of Standards, Washington, D.C. 1980.

[Yao 79] Yao, S. B. Optimization of Query Evaluation Algorithms. ACM Trans. Database Sys., Vol.4 (1979), 133-155.

[Yohe 79] Yohe, J. M. Implementing Nonstandard Arithmetics. SIAM Review, Vol.21 (1979), 34-56.

III

NETWORK OPERATING SYSTEMS: PERSPECTIVES AND HIGH LEVEL INTERFACE ISSUES

Stephen R. Kimbleton

National Bureau of Standards

1977

ABSTRACT

The emerging interest in Network Operating Systems (NOSs) reflects the needs for more effective user support, more efficient access, and better use of resources in a networking environment. Although often viewed as the network analog of host operating systems, this view proves inaccurate if NOS support is to be provided in the context of existing hardware and software. This report specifies alternative NOS views, identifies functional requirements, defines user groups, and observes that the two key support requirements are user-system and system-system interfacing. Issues which must be considered in each of these categories are discussed and related research activities are summarized.

1.0 INTRODUCTION

Computer communication networks, hereafter referred to as networks, consist of a collection of computers interconnected via a communications subnetwork or, simply, subnetwork and have been extensively investigated. (See Kimbleton [KIMBS 75] as an example survey.) Potentially, networks can more effectively satisfy the user's computing requirements through sharing of hardware, programs and data resources, as well as through more efficient use of systems through load leveling to balance time varying demands against time available capacity.

Presently, realizing this potential requires knowledge of the operating conventions, command languages, capabilities, and idiosyncracies of each of the systems being accessed. This already sizable information base must be augmented by knowledge of the subnetwork capabilities and the procedures for their uses which may be system-dependent. The collective information requirement is sufficient to deter all but the most resolute user and, in practice, tends to limit networking use to computer programmers and a very few other users.

To achieve widespread use of networking requires at least: offloading the burden of learning tedious details from the user; homogenizing the user's interface to heterogeneous systems; and standardizing key system-system interface requirements. The term Network Operating System

(NOS) has become common to describe the mechanism performing this offloading.

It is natural to view the objective of a NCS as providing, for a network, the analog of those functions provided by individual operating systems. Unfortunately, this objective, although intuitively reasonable, proves practically infeasible in general due to the large number of differing system commands and responses that must be accommodated.

The author would suggest that a higher level programming language (say FORTRAN) provides a better model. Thus, provided no errors are encountered, one can usually assume a FORTRAN view of the system. However, when errors are encountered, one may not be able to avoid the need to learn more about the detailed, low-level operation of the system. The designer of the processor for a given language is consequently faced with a tradeoff between the number of low-level details which can be masked and the cost of doing so.

The preceding observations can be summarized by stating that a NOS serves as a mediator between the user and the systems being accessed. Thus, the computing requirements of the user are satisfied by the accessed system and the NOS renders such access as easily as possible.

Although the general objective of a NCS is intuitively clear, the specifics require definition. In an earlier paper [KIMBS 76] the author described user objectives underlying NOS design. This report describes the functional requirements implicit in meeting these user objectives. In particular, it shows that these functional requirements can be divided into two categories: interfacing users to systems and interfacing systems to systems. The precise requirements of each category are detailed in sections three and four. Section five then describes some ongoing related research projects while section six presents concluding observations.

Finally, it should be noted that our discussion of user-system and system-system interface issues is restricted to high-level considerations. Specifically, their implementation assumes an underlying host-host communication protocol [CARRC 70], [CERFV 74] which, in turn, assumes some means for interconnecting hosts to the communication subnetwork [FOLTH 77]. Given these assumptions, it follows that our discussion is effectively independent of the underlying subnetwork technology which may be either packet- or circuit-switched or one of the hybrid technologies.

2.0 NETWORK OPERATING SYSTEM (NOS) PERSPECTIVES

This section describes: user requirements, functional components, implementation environment and primary implementation issues for a General Purpose Network Operating System (GPNOS). The prefix 'General Purpose' distinguishes between the collection of functions discussed in this report and those in other mission-oriented computing systems, which significantly overlap GPNOS functions and are often referred to as having NOS capabilities. Some of these related systems are discussed in section five.

2.1 GPNOS USER COMMUNITY

Characterizing the computing activities of a generic user provides one means for identifying the user community of a NOS. Such activities can be divided into three categories: i) accessing existing services; ii) construction of new services; and iii) construction of system-level mechanisms to support construction of new services.

Typically, item (i) corresponds to end-users, item (ii) to applications programmers, and item (iii) to systems programmers. However, a characterization in terms of function rather than job category is preferrable, since a given user may function in each of these roles during the course of a single terminal session. An appreciation of this definition requires an understanding of the scope of the term 'services'.

The services available on either an individual system or a network vary in complexity and range from text processing to data base querying to transaction processing. Thus, intuitively, a service corresponds to a debugged application or function provided by a computer system. This view stimulates an interest in measuring the performance of services rather than systems [ABRAMS 74]. Note that special cases of services include the ability to execute jobs and manipulate files as required to support payroll, accounts receivable, inventory, logistics, and compilation. (To appreciate the potential complexity of provided services, cf. the discussion of the National Software Works in section five.)

Easy access to network services requires that one be able to: i) freely access different hosts containing specific service components, ii) simultaneously access multiple hosts, iii) execute jobs on one or several hosts, and iv) manipulate files on a network-wide basis. Service constructors clearly require the functions afforded service accessors and, additionally, require access to high-level, nonprocedural mechanisms that support subnetwork communications and easy access to different systems. System implementors provide these nonprocedural mechanisms. Their provision, as would be expected, proves difficult because of the significant amount of low-level, system specific knowledge required for their effective construction.

Viewed from this perspective, the functional objectives of NOS are to support and simplify access to existing services and to expedite the construction of new services through simplifying communication among systems and between systems and users. The remainder of this section adds substance to this general statement.

2.2 GPNOS FUNCTIONAL COMPONENTS

A NOS providing ease of access to and use of systems, subnetwork, and services must provide four major functions:

o user interfacing,

o network job execution,

o network data support, and

o control.

We now identify the major objectives of each of these components. The attainment of these objectives for the network job execution and network data support components is then discussed in Sections 3 and 4 in the context of user-system and system-system interfaces.

2.2.1 USER INTERFACING -

The user interface provides the basic environment defining the interaction between the user and network accessible resources. A natural objective of this interface is to ensure that functionally similar software on different systems can be invoked via a single, common sequence of commands. Since users interact at either the command language level or the service level, the need for a common command language across systems is evident.

It is also natural to provide common versions of frequently used services such as editors. However, this proves to be a very difficult problem presently, since the organizations providing the services may be unwilling to provide a common interface, and the services may differ substantially. An alternative is to offload common functions onto support computers. This approach is easily feasible given a NOS environment and has the added advantage of clearly separating the service responsibilities of NOS implementors from those of the service providing hosts.

2.2.2 NETWORK JOB EXECUTION -

A network job consists of a collection of network job steps, each of which may be executed on a different system. As a result, the output of one job step may need to be migrated to another site for successive job steps. Thus, a means for initiating job steps, migrating output information, disposing of output results, and interacting with the user is required.

2.2.3 NETWORK DATA SUPPORT -

Network Data Support provides: i) a Network Wide Directory (NWD) for providing a uniform network-wide name space. The actual names used on the individual hosts may well differ due to differences in host naming conventions, but this should be transparent to the user. ii) a Common Command Language for File Manipulation (CCL/FM), and iii) a way to preserve the meaning of data being transmitted between systems.

Note that the Network Wide Directory is substantially different from the Local Host Directory (LHD), since the NOS does not have any responsibility for the mapping of the logical file onto physical storage. Moreover, for files consisting of one or a few record types, the NWD is also the repository of file descriptor information required to permit preservation of meaning in transmission across heterogeneous systems. Thus, the NWD description of a file must contain a file characterization suitable to support relocation of the file on an arbitrary system.

2.2.4 CONTROL -

The control mechanisms required by a GPNOS are extensive and include: i) an ability to control the impact of an individual user upon a host in terms of both resource utilization and the resources available to the user (resource addressability), ii) matching of resource requirements to resource availability required for load leveling and resource sharing, iii) network-wide accounting information, iv) generalized access control, v) advising the individual of the existence and accessibility requirements of services, and vi) balance of payments/flow of funds.

Although these issues are clearly of importance in ensuring GPNOS usability, determining requirements implicit in their support awaits the results of user experience. Accordingly, the general issue of control is not discussed further in this report.

2.3 GPNOS IMPLEMENTATION ENVIRONMENT

The difficulty of GPNOS implementation is critically affected by two factors: i) translation requirements, and ii) retrofit requirements. Translation requirements reflect the need to provide a common interface across heterogeneous systems at the command language and process execution level. Retrofit requirements reflect the need to add software and hardware to a system to upgrade it to be useful in a networking environment. The difficulty in meeting translation and retrofit requirements depends on the constraints imposed by the requirement that a GPNOS must interface with existing operating systems, utility programs, and applications programs. Thus, one can distinguish three levels of constraints in GPNOS implementation corresponding to: heterogeneous host systems, homogeneous host systems, and unconstrained implementation.

2.3.1 HETEROGENEOUS IMPLEMENTATION ENVIRONMENT -

Resource sharing is facilitated in proportion to the number of different resources made available through networking. As a result, translation support is required to support uniform access to heterogeneous systems. Furthermore, capabilities must usually be added to support networking requirements. Such capabilities extend those of individual systems and include Interprocess Communication (IPC) as well as network job execution, preservation of meaning in transmitting data, and support of host computer-to-network connection.

2.3.2 HOMOGENEOUS IMPLEMENTATION ENVIRONMENT -

In a homogeneous environment, translation requirements can be significantly reduced, if not eliminated, since the host Operating System Command Language (OSCL) provides a common language for interfacing users to hosts while the command language used to communicate with the subnetwork provides a suitable communications extension to the OSCL. The interface may prove to be somewhat ragged and, as a result, implementation of some higher level commands facilitating usage may prove desirable. Furthermore, mechanisms supporting cooperation across systems and connection of the host to the network are required just as in the heterogeneous case.

2.3.3 UNCONSTRAINED IMPLEMENTATION -

Eliminating the need to interface with existing software enhances both conceptual clarity and cleanness of implementation. The resulting implementations [FARBD 72],[MILLD 76] have provided significant insight into the basic architectural opportunities and requirements provided by an unconstrained implementation. Indeed, in such implementations, the user is effectively freed from the need to distinguish between 'local' and 'remote' items and may, instead, refer to objects in a uniform, location-independent manner.

Although unconstrained implementations are rich in insight, they are costly, since large amounts of existing software must be rewritten. These include individual host operating systems and applications software such as compilers. This problem can be reduced somewhat through a design which maximizes the interface with existing software; however, this constrains the implementation.

Unconstrained implementation of a GPNOS also eliminates upwards compatibility between existing applications and those made possible in a networking environment. As a result, the user is faced with a substantial rewrite requirement or, alternatively, an extremely prolonged changeover based on the natural obsolescing of running programs. Since emulation of earlier architectures still proves to be very popular, one cannot be too optimistic about an obsolescence based changeover.

In summary, we observe: i) unconstrained GPNOS implementations will be the most satisfactory from a user viewpoint but lack upwards compatibility and, as a result, will be extremely costly to implement, since significant rewriting of existing software is required, ii) homogeneous GPNOSs are easier to implement than heterogeneous GPNOSs, but limit the opportunity for resource sharing--a major rationale for GPNOS construction, and iii) heterogeneous GPNOSs constitute the most difficult implementation environment but maximize resource sharing and best accommodate both planned and existing systems.

2.4 GPNOS IMPLEMENTATION ISSUES

The preceding discussion has described the user community for a GPNOS, identified the required functional components, and discussed the environmental issues affecting the complexity of GPNOS implementation. Given this context, it is evident that the primary implementation issues can be divided into two major categories:

- o interfacing users to systems, and
- o interfacing systems to systems.

These issues will be discussed in detail in the following two sections.

3.0 INTERFACING USERS TO SYSTEMS

Interfacing users to systems implicitly establishes a virtual environment for the user consisting of the user's view of each of the accessible systems together with any mechanisms which may be provided to coordinate activities across systems. It follows that the salient issues are: identification of the features which are to be provided and consideration of the 'tightness' of this virtual environment (i.e., extent to which the user need only consider the virtual environment and may ignore the underlying 'real' environments). Although definitive examination of this latter issue remains to be done, our subjective evaluation was expressed earlier in the statement that the interface will be similar to that provided by a high-level programming language.

Interfacing users to a system requires three major support functions: i) a Network Wide Directory (NWD) listing the objects contained in the user's virtual environment, ii) an Access Control Mechanism (ACM) identifying any restrictions imposed on the user in accessing objects identified in the NWD, and iii) command language mechanisms for manipulating files and running programs listed in the NWD.

3.1 NETWORK WIDE DIRECTORIES

Determination of the precise information to be contained in the NWD is, at present, an unsolved problem. However, the underlying objective is to provide those attributes of an object required to facilitate its usage in a networking environment. This requires at least:

- o naming transformations,
- o name resolution,
- o existential correlation, and
- o context mapping.

We now discuss each of these items in turn.

3.1.1 NAMING TRANSFORMATIONS -

Users and systems have differing requirements in naming objects such as programs or files. System requirements are usually arbitrary (e.g., names will be at most nine alphanumeric characters long with a maximum of six characters to the left of a period and three to the right) and reflect historic concerns with storage costs and computer architecture. User requirements are more closely related to the need to identify the object's properties and the desire to enter only the minimal amount of information necessary to support interactive access. Thus, file names are permitted to contain information indicating the anticipated use of the file, such as source program, object program, and programming language, and means are provided on some systems (e.g., the TENEX operating system for the DEC PDP-10) which permit the user to enter only enough characters to uniquely identify the file name and then, through pressing the 'escape' key, to have the system provide the remaining characters in the file name.

In view of the different naming requirements of users and systems, a user-oriented NOS should accomodate user-oriented naming conventions. This implies that mechanisms for mapping between user names and those syntactically acceptable to the system must be provided. However, since different systems have different syntactically acceptable names, such mechanisms must be provided anyway in a heterogeneous environment. As a result, a requirement for translating names among systems cannot be avoided short of the obvious approach of requiring everyone to use names acceptable to all systems--surely very undesirable in view of NOS objectives.

Network operating systems, in addition to providing user-oriented naming conventions, should also provide a means to provide subdirectory capabilities for systems lacking this option. Such subdirectories provide two major features: a means of organizing directories into logically related subdirectories, and a means of confining access to files within a subdirectory.

Since, from a NOS viewpoint, names are logical entities, it follows that they can be arbitrarily rearranged into groups or subdirectories. To confine program access requests to a subdirectory only requires that any running program be given its own directory at run time. Finally, granting of access privileges to subdirectories follows easily, since the access is at the logical level and requires the use of a transformation function or mapping to yield actual read/write/execute rights.

Although user-oriented naming conventions are feasible in a NOS environment, some attention must be given to the problem of mapping network names into local host names.

3.1.2 NAME RESOLUTION -

Name resolution supports the mapping of NOS names into local host names. Since the local host names are only names and are not intended to provide context information about the named objects, very simple maps based on a serial assignment of alphanumeric characters may be adopted. Thus, the first file based on some ordering such as order of creation in a directory could be named A, the second B, the twentyseventh AB, and so forth.

Although the preceding approach solves the problem of mapping network names into local host names and, implicitly, the converse, this only satisfies user-oriented requirements. Making names known to programs remains.

Names can be made known to programs at two distinct times: the time at which a (run) command is issued, or dynamically during program execution time. Handling the first problem is reasonably straightforward, since the user-oriented names can be mapped into local host names by the command language processor provided as part of the NOS.

Handling dynamically generated names is a difficult problem for which only interim approaches have been identified. Such approaches can be divided into two categories: trapping and prepositioning.

Trapping is based on the philosophy that whenever a name request is generated, a trap is forced to software which determines whether the referenced object is local or remote and, if remote, forces the necessary actions to make the network object local with the name anticipated by the program generating the trap. Such an approach is implicit in the implementation of RSEXEC [THOMR 73,75] and JSYS traps in the TENEX operating system.

Prepositioning is possible when one can anticipate all the objects that can be referenced by a program. Specifically, one can develop 'templates' which identify the object and its relationship to the program being run. As a result, the command language processor can arrange to take any necessary actions to ensure addressability when a reference occurs.

The preceding approaches tacitly assume that a program is neither created nor modified to explicitly recognize the existence of the network. If this constraint is eliminated, substantially simpler name resolution can be achieved through forcing references via subroutine CALLs. Note that this is the approach followed in supporting communication between programs and Database Management Systems (DBMSs) and offers many advantages. Specifically, local versus global distinctions can be reduced and substantially more powerful capabilities can be provided to accommodate and resolve differences across systems.

3.7.3 EXISTENTIAL CORRELATION -

Linking files or directories together with programs at the time of implementing the programs serves to establish correlations among named objects. It follows that protective mechanisms are required to ensure that object modifications do not adversely affect others dependent on the object. Although this requirement is desirable in the context of an individual host, it is significantly more important in a networking environment. This reflects the fact that remote users are less likely to be aware of the intention to modify accessible objects. As a result, such modifications should be supported by the NWD through provision to accessors of information regarding the types and dates of modification. This implicitly requires some cooperation by the modifier or, alternatively, a version capability precluding modification of any version of a program having (remote) network access capabilities.

3.7.4 CONTEXT MAPPING -

Network utilization implies that the system accessed by an individual user may change with time during the course of a single terminal session. As a result, there is a need to facilitate such change. This implies the requirement for semiautomatic capabilities which support transitions among systems through migration of files, directory modifications, and so forth. Determination of the precise requirements implicit in context mapping promises to be an exciting research area.

3.2 ACCESS CONTROLS

Within the context of an individual computer system, access controls have traditionally been rather primitive. For example, delineation of read/write/execute privileges limited to the owner, the group to which the owner belongs or the universe of users is often regarded as acceptable. In contrast, substantially more sophisticated requirements exist in a networking environment.

Network level access controls can be implemented with varying degrees of granularity. Thus, although the file is usually the level at which access controls are provided by a host operating system, Network Operating Systems require finer granularity extending to the individual data element to support a closer matching of user access rights to data element access controls. Achieving network level access controls which are more sophisticated than individual host access controls appears possible, since it seems feasible to implement the NWD as a protected subsystem supporting control of remote access to its named objects.

AD-A102 776 NATIONAL BUREAU OF STANDARDS WASHINGTON D C INST FOR--ETC F/G 9/2
XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER.(U)
JUN 81 S R KIMBLETON, P S WANG, E N FONG F30602-77-C-0066
UNCLASSIFIED RADC-TR-81-123 NL

3 OF 5
AD
A102776

3.3 COMMON COMMAND LANGUAGES

Command languages for an individual host [UNGEC 74] provide users with a means to manipulate files, run programs, and make files known to programs. Network-level common command languages must provide the same basic mechanisms. Satisfaction of this requirement in a networking environment clearly requires translation of commands into those acceptable to the host on which a command is being executed.

The difficulty of implementing a common command language increases with the spectrum of commands provided. From a NOS viewpoint, however, the number of required commands is significantly less than the number required on an individual host. This reflects the fact that the NOS deals with logical entities. As a result, it need not have commands for explicitly positioning objects on secondary storage devices or for determining the memory hierarchy properties of objects.

The preceding comment suggests a slightly philosophical statement paralleling our comment that NOSs are similar to high-level programming languages. In particular, the user with trivial computing requirements can probably remain peacefully oblivious to the network, while the user with substantial requirements will probably require escape capabilities providing direct access to host features. For users not falling into either of these two categories, it seems feasible to satisfy computing requirements through the NOS and to restrain object manipulations to the logical level, as opposed to the physical or memory hierarchy level.

Given the preceding approach, the number of actual commands required to support a network wide common command language is minimal. We require commands for network job execution which support executing a named file, making network file names known to the program and establishing the correspondence between program generated files and their desired network names. Assuming that 'no setup jobs' is the only job class of interest, it turns out that only one command is required.

In addition to the network job execution command, a series of commands are required to manipulate files. Since these are logical level commands, their number is small, and an initial implementation using the NBS Network Access Machine [ROSER 75] based on fewer than twenty separate commands is underway [FITZM 77]. Finally, directory-like commands are required to permit the user to find out the status of the files. Determination of the spectrum of such commands is open, reflecting the fact that identification of the precise properties to be provided by the directory is still at a research stage.

Having identified the requirements implicit in interfacing users to systems, we now consider those of interfacing systems to systems.

4.0 INTERFACING SYSTEMS TO SYSTEMS

Two key implementation issues must be considered in interfacing systems to each other: Interprocess Communication (IPC) and Remote Record Access (RRA).

4.1 INTERPROCESS COMMUNICATION

Subroutine calls are a common way of invoking utilities or programs prepared by others within an individual system. As a result, there has been relatively little need for most users or applications programmers to use interprocess communication which provides the mechanism to support communication between two executing programs or processes. This is reflected in the absence or limited implementation of IPC capabilities within many existing computer systems and the difficulties in the access, use and sophistication of any provided capabilities.

In contrast with the single site case, effective use of networking capabilities requires IPC. This reflects the fact that the only active entity within a computer system (i.e., the only entity which can control devices, issue device requests, etc.) is a process. However, communication between systems basically requires communication between active elements or processes. Thus, the fundamental nature of the IPC requirement in a networking context.

Having established the need for IPC capabilities we now consider three fundamental aspects:

o levels of sophistication,

o existing capabilities, and

o future needs.

IPC LEVELS OF SOPHISTICATION

Interprocess Communication can be provided at four increasingly sophisticated levels:

o endpoint interactions,

o CALL/RETURN,

o message based, and

o synchronization.

Endpoint interactions describe the sort of communication between processes which only occurs at their initiation or termination. Job Control Languages typically provide such capabilities and, as is evident from our earlier discussion, are clearly required to support Network Job Execution. This form of IPC might well be regarded as degenerate, since process dependencies can only be operative at initiation/termination time and not during execution.

IPC via a CALL/RETURN mechanism constitutes the simplest mechanism supporting run time dependencies between processes. The simplicity reflects the fact that, at any point in time, precisely one of the CALLer or CALLee processes is active. This form of IPC can therefore be regarded as a natural analogue of a subroutine CALL/RETURN mechanism. The natural disadvantage reflects the significant amount of time required to service an IPC request in a networking environment due to subnetwork delays.

To permit concurrent execution of processes, the next level of sophistication is message based IPC, in which one process can send a message to another which is either WAITing or active while continuing to execute. This clearly supports concurrent and coordinated execution of several processes on different systems. However, any explicit coordination capabilities must be explicitly user-programmed.

Issues implicit in providing message based IPC have been extensively considered [SUNSC 77],[WALDD 72],[ZUCKS 77] in the context of the UNIX operating system (UNIX is a trademark of Eell Laboratories, Inc.) for the PDP-11 series computers. Their effective utilization requires both an ability to permit multiple senders to a given process and to permit a process to examine the collection of waiting messages to determine the identity of the senders.

Intrahost IPC support of synchronization and coordination has been widely studied [BRINP 73] in the context of buffers, sections of code, and so forth. Support of such capabilities at the network level clearly requires the existence of intrahost capabilities which, at the present time, are not uniformly available. However, since the objects whose access is being synchronized must reside within one host or another, it is not clear that a strong requirement exists for this type of IPC at the network level.

4.7.1 EXISTING CAPABILITIES -

At the present time there are no existing general purpose protocols supporting Interprocess Communication on the ARPANET. Thus, the user is faced with a requirement for constructing any required intersystem capabilities and using those existing within any given system to provide intrasystem capabilities.

Although general purpose IPC capabilities are lacking, as evidenced by their omission from the ARPANET PROTOCOL HANDBOOK revision of [illegible] April 76, substantial work is underway in this area to support such major network based projects as the National Software Works (NSW), which has implemented a message based approach to network IPC providing commands for SENDing, RECEIVEing and STATUS determination [THOMR 76].

4.7.2 FUTURE NEEDS -

At the present time, substantial discussion has been devoted to considering the relative merits of CALL/RETURN mechanisms which operate in a manner analogous to a subroutine call and message based approaches which transmit a message to a foreign process upon request but permit the transmitting process to continue execution. Although it is too early to provide a definitive comparison of the relative merits of these two approaches, some comparative comments can be provided.

A key consideration in comparing these two approaches is the adopted viewpoint of network capabilities. Two extremes can be identified. The first corresponds to a viewpoint in which effectively autonomous computers and programs written for serial execution on a given host are subsequently provided with a means for network access. Under these circumstances, upon the issuance of a CALL, there is little that an individual program can do other than wait for the RETURN with the result that it is appropriate for the program to enter the WAIT state pending the return.

The second viewpoint is that common to construction of the ARPANET communications subnetwork routing algorithm. In particular, this algorithm is implemented as a distributed computation in the collection of packet switches within the communication subnetwork. Specifically, each IMP determines the optimal successor IMP for each destination based upon information generated internally as well as information forwarded from each adjacent IMP. If the information is forwarded as expected, appropriate tables are updated and, if not, the increasingly out-of-date information from the last successful transmission is used. Note that, the design and implementation of the algorithm is predicated upon the possibility of error. As a result, use of a message based approach is in order. In particular, the transmitting IMP makes no assumptions regarding the status (up/down) of the receiving IMP which, in turn, is programmed to operate without requiring successful receipt of information from the transmitting IMP.

It is clear that these two viewpoints of networking differ substantially in their support requirements. As a result, it seems that a 'broad majority' consensus cannot be achieved until additional evidence has accrued indicating the generic nature of networking requirements in support of information processing applications.

Although the accumulated weight of evidence does not conclusively indicate which of the two discussed approaches is preferrable, it is clear that the message based approach is more general in the sense that given its existence, superimposition of a CALL/RETURN approach is straightforward. As a result, from a systems building viewpoint, selection of the message based approach seems more desirable.

Given a suitable network wide IPC mechanism, one can support execution of network jobs whose job steps are distributed over several systems. Moreover, existing ARPANET protocols permit transmission of files among systems and thereby support implementation of a file-level common command language. We now turn to consideration of the problems implicit in providing program access to remote records.

4.2 REMOTE RECORD ACCESS

In processing heavily compute bound jobs it is customary to bring the data to the job. For heavily I/O bound jobs one normally brings the program to the data. In the intermediate cases one may not be able to ensure that program and data are colocated. Accordingly, a means for allowing a program to access remote data at the record level is required.

Support of program access to remote records requires at least the following capabilities:

- o record selection,
- o record translation, and
- o record transformation.

To understand the implications of these functions in the context of remote record reading/writing, we agree to call the host bearing the file containing the remote record the Data Host (Dhost) and that bearing the accessing program the Process Host (Phost).

We assume that the Phost issues a request for a record located on the Dhost; we further assume that the desired record is indicated by either a unique key if random access techniques are being supported or, alternatively, by the keyword 'next' if sequential access is being used.

Given that a request has been received by the Dhost, the record selector is responsible for retrieving the record and transmitting this record to the record translator.

The record translator is responsible for ensuring the preservation of record meaning in transmitting the record across systems. Specifically, this requires preservation of the logical record structure as well as preservation of data element type and, for arithmetic data elements, maintenance of precision. A specific implementation approach for obtaining these objectives is described by Wood [WOODH 77]. Note that preservation of logical record structure may require significant amounts of 'bit manipulation', since, for example, some systems/languages store matrices by rows while others store matrices by columns.

Record transformation is required in order to match the information transmitted to the needs of the receiver or to ensure proper protection of sensitive information. Such transformation affects the logical structure of the record through one of three basic transformation types: logical, arithmetic or string.

Logical transformations such as 'and' or 'or' generate boolean strings resulting from the bit-by-bit anding and oring of two successive strings. Arithmetic transformations such as: +,-,/,x act as would be expected. String transformations can potentially be quite complex as is evidenced by the capabilities of string manipulation languages. Initially, a concatenation capability seems desirable. Implementation of a record transformation capability is also described by Wood [WOODH 77].

It should be noted that translation and transformation as discussed above are applied at the level of individual records. It follows that this approach provides an alternative means for achieving file translation/transformation [CERFV 72, SHUNC 76] at, perhaps, some cost in efficiency.

Although, as is intuitively evident, significant additional work is required to provide a remote record access capability, the preceding does indicate the major conceptual requirements and a more extensive discussion of actual implementation factors can be found in the cited reference. We now discuss some currently existing or soon to exist systems in the context of these requirements.

5.0 NOS RELATED PROJECTS

This section summarizes information on four major NOS related projects: RSEXEC, NSW, ACCAT, and XNOS. A more extensive discussion of the first two projects is given by Forsdiek [FORSH 77] and the references cited therein.

5.1 RSEXEC

The first NOS-related activity in an ARPANET context was the Resource Sharing Executive (RSEXEC) [THOMR 73] implemented for the collection of PDP-10 computers running the TENEX operating system within the ARPANET. RSEXEC provided a Network Wide Directory, a common file manipulation capability, and a RUN command which would force the migration of remote files to the site at which a program was to be run. In addition, utilities for determining status were also provided.

Since RSEXEC was implemented for a homogeneous collection of systems, translation mechanisms were not required. Moreover, interhost IPC and Remote Record access are also lacking. However, it is possible for other systems such as Multics to implement the RSEXEC protocol and, thereby, to extend this protocol to a heterogeneous environment provided the user supports any required translation.

The ability to extend RSEXEC in the manner described depends upon its treatment of references to files. In effect, such references result in a trap to software which determines if the file is local or remote and, if remote, arranges for its retrieval. The same process could clearly be applied to files containing structured and typed records provided that the appropriate record descriptions and translation software were available.

5.2 NATIONAL SOFTWARE WORKS

Users use computer communications capabilities to achieve certain goals. Often the achievement of these goals can be obtained in a fairly direct manner through running a job or querying a database. In some cases, however, a collection of related and integrated activities is required as is true for the production of software.

The National Software Works [NSW 76] is intended to provide a software production capabilit' containing the most sophisticated tools available to assist in this objective. Since these tools are not all available on the same machine or vendor line, use of networking is in order to provide the user with access to the collection. Moreover, since the needs of the user are known, a design facilitating their use can be employed.

Provision of a software production capability requires four major components: i) a component controlling the interaction of a software tool with its local host system (Foreman), ii) a component supporting the interaction between the user and the tool (Front End), iii) a component for controlling the overall use of NSW resources, for providing access controls, and for arranging communication paths among systems (Works Manager), and iv) a file package which is responsible for file movement and has some translation capabilities. The collection of these four components which, potentially, can themselves be distributed across systems, together with the supported tools, constitutes the NSW.

The objective of the National Software Works is to provide a software production capability; the consequent desirability of making hosts transparent to the user is evident. As a result, when the user issues a command to run one of the provided software production tools (programs), it is the NSW that determines the locations at which the tool is available, decides which location is 'best', and connects the user to the tool. Thus, conceptually, the NSW provides support similar to that provided by an unconstrained NOS implementation for the user interacting with the collection of constitutent tools.

NSW clearly provides a common command language and Network Wide Directory together with a run command which is effectively limited to the provided tools. In addition, it provides a sophisticated, message based interhost IPC capability [THOMR 76]. Remote record access is not provided, some translation is likely to be provided at the file level, and record transformation is not provided.

5.3 ADVANCED COMMAND AND CONTROL ARCHITECTURAL TESTBED

Data is increasingly recognized as one of an organization's most valuable resources [DATEC 75]. The increasing availability of query languages to provide online user- access to Database Management Systems (DBMSs) supports the need by high-level management for fast response to questions arising in the course of strategic planning. (Following Anthony [ANTHR 65] we distinguish between operational control, managerial control and strategic planning components of information processing. Cperationally, as one passes from the first to the last component, the volume of data and predictability of its use decrease.)

Networking clearly promises enhanced strategic planning support, since it provides access to geographically dispersed data bases. A major impediment is the need for the user to become proficient in the systems being accessed and to be knowledgeable about the structure of the accessed data base. Clearly, much of this burden could be offloaded to an 'intelligent' system, thereby facilitating the strategic planning of the organization.

In partial recognition of the importance of the cited problem and the rewards implicit in its solution, research is currently being conducted in providing an intelligent frontend to a data base management system and, ultimately, to a networked collection of such systems as described by Morris [MORRP 77] and Hendrix [HENDG 77]. Thus, the user is permitted to express queries in a somewhat structured format which are then transformed into a series of sub-queries against the DBMS(s). The results are then aggregated and returned to the user.

Since ACCAT is still in the initial implementation stages, it is premature to attempt to determine the NOS-related support mechanisms which it will provide. However, it is clear that the networked DBMS support it provides complements the support provided in other environments discussed above, and will provide insight into the sophistication of the interface which can be made available to the user in interacting with data in a networking environment.

5.4 XNOS

A General Purpose Network Operating System provides a powerful mechanism for interfacing users to systems and systems to systems. Thereby, the effectiveness and efficiency of users of these systems is substantially enhanced. To better establish the precise nature of the required interfaces, a joint RADC/NBS project was conducted at the National Bureau of Standards to develop an Experimental Network Operating System (XNOS). The XNOS has successfully demonstrated the feasibility of a common command language for heterogeneous systems [FITZM 77], a Remote Record Access capability [WOODH 77], and an outboard IPC mechanism [KIMBS 77].

6. CCNCLUDING REMARKS

The objective of this paper was the provision of a perspective ɔn NOS requirements together with a discussion of the major user-system and system-system issues which must be considered in NOS implementation. As we have seen, the difficulties of providing these interface capabilities are dependent upon the environmental constraints which must be considered. However, even for the evidently most difficult case -- retrofitting in a heterogeneous environment, NOS interface requirements can be cleanly structured into a remarkably small number of categories.

It is premature to attempt to assess the extent to which Network Cperating Systems can be regarded as a 'success' and the rate at which GPNOS software will appear. Nevertheless, in view of the preceding structural observation, it is our considered conclusion that success is highly likely, that the problems in software construction, if suitably structured, are less than would be anticipated, and that the resulting system overhead can be held to an acceptable level.

REFERENCES

[ABRAM 74] Abrams, M. D., "Measuring Service Delivered in Interactive Computing," Proceedings Second Jerusalem Conference on Information Technology, August, 1974, pp. 747-754.

[ANTHR 65] Anthony, R., PLANNING AND CONTROL SYSTEMS: A FRAMEWORK FOR ANALYSIS, Division of Research, Graduate School of Business Administration, Harvard University, Boston, Mass., 1965.

[BRINP 73] Brinch Hansen, P., OPERATING SYSTEM PRINCIPLES, Prentice Hall, Englewood Cliffs, New Jersey, 1973.

[CARRC 70] Carr, C. S., S. D. Crocker, and V. G. Cerf, "host-host Communication Protocol in the ARPA Network," Proc. AFIPS 1970 Spring Joint Computer Conference, Vol. 36, AFIPS Press, Montvale, New Jersey, 1970, pp. 589-597.

[CERFV 72] Cerf, V. G., E. F. Harslem, J. F. heafner, R. M. Metcalfe, and J. E. White, "An Experimental Service for Adaptable Data Reconfiguration," IEEE Transactions on Communications, Vol. COM-20, No. 3, June 1972.

[CERFV 74] Cerf, V. G., and R. E. Kahn, "A protocol for packet network intercommunication," IEEE Transactions on Communications, May 1974, pp. 637-648.

[DATEC 75] Date, C. J., AN INTRODUCTION TO DATABASE SYSTEMS, Addison Wesley, 1975.

[FARBD 72] Farber, D. J. and K. Larson, "The System Architecture of a Distributed Computer System-the Communications System," Proc. Symposium on Computer-Communication Networks and Teletraffic, Polytechnic Press, Polytechnic Institute of Brooklyn, New York, April 1972, pp. 21-27.

[FITZM 77] Fitzgerald, M. L., "Common Command Language for File Manipulation and Network Job Execution," in preparation.

[FOLTH 77] Folts, H. C. and I. W. Cotton, "Interfaces: New Standards Catch Up with Technology," Data Communications, June, 1977, pp. 31-40.

[FORSH 77] Forsdick, H. C., R. E. Schantz, and R. H. Thomas, "Operating Systems for Computer Networks," BBN Report No. 3614, Bolt Beranek and Newman, Cambridge, Mass., 1977.

[HENDG 77] Hendrix, G. G., "Lifer: A Natural Language Interface Facility," Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, pp. 196-201.

[KIMBS 75] Kimbleton, S. R. and G. M. Schneider, "Computer Communication Networks: Approaches, Objectives and Performance Considerations," in Computing Surveys, September 1975.

[KIMBS 76] Kimbleton, S. R. and R. L. Mandell, "A Perspective on Network Operating Systems," Proc. 1976 National Computer Conference, AFIPS Press, Montvale, New Jersey, Vol. 36, 1976, pp. 551-559.

[KIMES 77] Kimbleton, S. R., "Network Operating Systems: Issues, An Approach, and Some Conclusions," in preparation.

[MILLD 76] Mills, David L., An Overview of the Distributed Computer Network, University of Maryland, College Park, Maryland, 1976.

[MORRP 77] Morris, P. and D. Sagalowicz, "Managing Network Access to a Distributed Database, "Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, Berkeley, California, pp. 58-67.

[NSW 76] "National Software Works," Semi-Annual Technical Report, Massachusetts Computer Associates, Inc., Wakefield, Massachusetts, 1976.

[ROSER 75] Rosenthal, Robert, "Accessing On-Line Network Resources with a Network Access Machine," IEEE Intercon 1975, IEEE, New York, 1975.

[SHUNC 76] Shu, Nan C., Vincent Y. Lum and Barron C. Housel, An Approach to Data Migration in Computer Networks, IBM Research Laboratory, 1976, IBM Research Report RJ1703.

[SUNSC 76] Sunshine, Carl, "Factors in Interprocess Communication Protocol Efficiency for Computer Networks," 1976 NCC Proceedings, Vol. 45.

[SUNSC 77] Sunshine, Carl, "Interprocess Communication Extensions for the UNIX Operating System: I. Design

Considerations," Report R-2064, The RAND Corporation, Santa Monica, Ca., June, 1977.

[THOMR 73] Thomas, R. H., "On the Design of a Resource Sharing Executive for the ARPANET," Proc. AFIPS 1972 National Computer Conference, Vol. 42, AFIPS Press, Montvale, New Jersey, 1973, pp. 155-164.

[THOMR 75] Thomas, R. H., "JSYS Traps--A Tenex Mechanism for Encapsulation of User Processes," Proc. 1975 National Computer Conference, AFIPS Press, Montvale, New Jersey, 1975, pp. 351-360.

[THOMR 76] Thomas, R. H. and S. C. Schaffner, "MSG: The Interprocess Communication Facility for the National Software Works," BBN Report No. 3483, BBN, Boston, Massachusetts, 1976.

[UNGEC 74] Unger, C., COMMAND LANGUAGES, North Holland/American Elsevier, New York, 1974.

[WALDD 72] Walden, D. C., "A System for Interprocess Communication in a Resource-Sharing Computer Network," Communications of the ACM, Vol. 15, No. 4, April 1972, pp. 221-230.

[WOODH 77] Wood, H. M. and S. R. Kimbleton, "Remote Record Access: Requirements, Implementation, and Analysis," in preparation.

[ZUCKS 77] Zucker, S., "Interprocess Communication Extensions for the UNIX Operating System: II. Implementation," Report R-2064/2, The RAND Corporation, Santa Monica, Ca., June, 1977.

IV

THE NETWORK OPERATING SYSTEM XNOS IN RETROSPECT

Pearl S-C. Wang

Stephen R. Kimbleton

National Bureau of Standards

June 1980

ABSTRACT

XNOS is an experimental network operating system developed at the National Bureau of Standards. During the three years since the beginning of the effort, two versions of the system were built. This paper is a reflection on the system: its goals, its architecture, its implementation environment and the lessons we have learned from the building and rebuilding of XNOS.

The various issues involved in the construction of Network Operating Systems are described in detail within the context of this specific system. The design and implementation choices for both versions of XNOS are given and compared as to their strengths and weaknesses. Some conclusions are drawn concerning the appropriateness of the XNOS approach to the implementation of network operating systems.

1. INTRODUCTION

The first version of the network operating system XNOS was constructed on the UNIX (*) [Ritchie 74] system running on the PDP-11/45 computer at NBS in 1978 [Kimbleton 78]. This version provides network-wide file service in the form of a network-wide directory system (nwds) and a common command language (ccl) for file manipulation as well as a remote job entry (rje) facility. It also supports, as a separate utility (i.e. not part of the ccl), the capability of transferring structured data between dissimilar network hosts in a meaning-preserving way, through the provision of 'remote record access', or 'rra', service [Wood 80].

With the completion of the implementation of XNOS in April 1978, it became apparent that the basic approach was undesirable. First, XNOS was not developed with reliability, in any realistic sense, in mind. It uses terminal emulation to interact with remote systems. This implies that the handling of remote host malfunctions will be extremely difficult, and in many instances, highly unsatisfactory. In addition, the 'blocking i/o' feature of UNIX [Ritchie 78] means that it is impossible for XNOS, given its Network Access Machine (NAM) basis [Rosenthal 78],

(*) UNIX is a trademark of Bell Laboratories.

to provide its users with run-time control of the system's operation, since during the execution phase of XNOS (when the NAM macros are being expanded [Fitzgerald 78a]), the program waits and will not respond to terminal signals until its operations are completed.

In October 1979, a second implementation of the system was undertaken. This version, XNOS-II, also runs on the PDP-11/45 UNIX system at NBS. It supports essentially the same network-wide user services as the original XNOS (that is, the nwds, ccl, rje and rra facilities), but in a somewhat improved fashion [Wang 80a]. It also contains significant enhancements, especially in the area of system management functions. Detailed discussions of these user-level differences and enhancements are given in the later sections of this report.

Apart from the above-mentioned differences, which are visible to XNOS users, a far more important difference lies in the internal structuring of the two versions of the system. The second implementation (XNOS-II) entirely abandons the NAM approach of the first version (XNOS). Instead, it is constructed directly on top of the UNIX system and the ARPANET [Frank 72] host-to-host and initial connection protocols [Feinler 78] as provided by the UNIX ARPANET interface program (NIP). In other words, it is built from the 'Network UNIX' system [Chesson 75].

The basic architecture of UNIX makes it easy to write, test and debug XNOS-II programs during the various stages of the development of the system. The ability of UNIX processes to create new processes and execute other programs make the incorporation of many user-friendly features very easy. For example, instead of interpreting the user's command string, by forking off a child process which executes the UNIX 'shell' with the original command string as input, we were able to provide the user with an interface to the UNIX from within XNOS-II in a very easy fashion. The network UNIX is also noteworthy in that local (i.e., baseline UNIX) and network functions are integrated in a natural way. Thus, interfaces between XNOS-II and the NIP are clean and simple.

However, the UNIX implementation basis has several disadvantages as well. Almost all of these difficulties stem from the fact that UNIX is not a 'real time' system [Ritchie 78] whereas XNOS-II needs to perform many real time functions. The most important defect of UNIX, in so far as XNOS-II implementation is concerned, is its lack of asynchronous i/o facilities. XNOS-II (as well as other types of networking software) needs to initiate i/o on several (logical) network channels and delay until the operation is complete on only one of them. This cannot be achieved straightforwardly in UNIX. Additionally, UNIX is also deficient in its interprocess communication capability. For example, the handling of exceptional conditions, e.g., the 'abort' signal from the terminal, is awkward and not very reliable. This is because UNIX only provides very limited and rudimentary ways for processes to communicate special messages to each other (cf. the 'signal' system call [Thompson 75]).

This report discusses the knowledge gained in the design and redesign of the system. Section 2 summarizes our goals and describes the various XNOS-II design decisions. Section 3 justifies these decisions and compares them with the original XNOS approach. Strengths and limitations of XNOS-II are discussed in Section 4, from both the user's and the implementer's point of view. Section 5 summarizes our design and implementation experiences and concludes with some retrospective remarks.

2. DESIGN GOALS

The goals of a Network Operating System (NOS), although discussed at length previously by one of us [Kimbleton 78], and also by others [Thomas 78], deserve a brief review. The following subsection presents a summary of the design goals and issues for NOSs in general, and for the NBS XNOS/XNOS-II system in particular.

2.1 Integration of Individual Host Services

Although many login and file transfer network services provide the communication support mechanism among geographically dispersed, heterogeneous computers, they do not alleviate the effects of differences across systems. A Network Operating System (NOS) eliminates these differences to a large degree. A NOS is the collection of software and supporting network protocols that integrates the o/s facilities of a diversity of interconnected but autonomous computers. The XNOS-II provides a computing environment that allows the various computer resources existing in a network to be used together in a convenient and cost-effective manner.

XNOS/XNOS-II is an attempt to organize and unify the o/s facilities of individual hosts to achieve a general purpose computing system for the larger, network-wide user community. The aforementioned report [Kimbleton 78] contains detailed discussions of the design rationale leading to the decision that XNOS/XNOS-II should be a centralized, network operating system. 'Centralized' describes the manner in which global resource allocation decisions are made--namely, unilaterally in the Network Interface Machine (NIM). 'Network' refers to the fact that local resources are controlled and managed by the individual hosts without being affected, in any way, by its participation in XNOS/XNOS-II (*).

That is, we have chosen to offload all NOS support functions onto a separate computer, the NIM, which deals with individual hosts on behalf of the XNOS user. This minimizes the impact on the existing network hosts, facilitates developmental and maintenance activities through centralizing these functions, and lowers operational costs by reducing the number of support personnel.

Of course, these advantages are obtained at a cost. Firstly, there is the need to maintain an additional machine, and secondly, without the replication of NIMs, the system will be vulnerable to failures of the centralized NIM site. Thus we are sacrificing some of the robustness of the system in order to gain ease of implementation and maintenance. A second essential decision is that

----

(*) The terminology used here agrees with that of others, see, for example, Thomas [Thomas 78], Forsdick [Forsdick 78] and Jensen [Jensen 80].

interactions between the NIM and individual hosts should be carried out in the mode of an unprivileged, time-sharing user, i.e., the CLI (command language interpreter) and NIP. This preserves host autonomy at the expense of system performance, since the resource allocation strategies adopted by the individual sites may sometimes be in conflict with global needs.

An additional advantage of building on top of existing host o/s's (i.e., the 'meta-system' approach of Thomas et al. [Thomas 78]) is that it generally requires less effort for its implementation than the 'building-from-scratch' or 'base-line' approach [Thomas 78]. Much of the existing individual o/s software can be retained to support the network operating system and the implementation effort can concentrate on augmenting these remote o/s functionalities to support the network environment.

In total, we are given the system objective of interconnecting separate computer systems to provide a common user interface with attendant decisions concerning alternatives such as what kind of system (local, network or global) to build and where (centralized or decentralized) control should reside. We have chosen the network approach with centralized management of all global functions and offloading of these functions onto a separate, special-purpose machine--the NIM. Furthermore, XNOS/XNOS-II adopts the meta-system approach for its implementation, with network local o/s interface performed at the terminal- or NIP-user level.

Since the advantages and disadvantages of this particular design choice have been given elsewhere [Kimbleton 80], we shall omit such considerations here. Rather, we shall proceed with a discussion of the XNOS/XNOS-II application environment and some of the design trade-offs that we have made in terms of specific XNOS/XNOS-II functions and facilities.

2.2 The XNOS Application Environment

Characterizing the computing activities of a generic user provides us with a means to identify the application environment of a system. Such computing activities can be divided recursively into three categories: i) accessing existing services; ii) construction of new services; and iii) construction of system level mechanisms to support construction of new services.

Typically, item (i) corresponds to endusers, item (ii) to applications programmers, and item (iii) to systems programmers. However, a characterization in terms of function rather than job category is preferrable, since a given user may function in each of these roles during the course of a single terminal session. An appreciation of this definition requires an understanding of the scope of the term 'services'.

The services available on either an individual system or a networked collection of systems vary in sophistication and range from text processing to data base querying to real-time data collecting. Thus, intuitively, a service corresponds to an application or function provided by a computer system. This viewpoint leads to an interest in measuring the performance of services rather than systems [Abrams 74]. Note that special cases of services include the ability to execute jobs and manipulate files as required to support payroll, accounts receivable, inventory, logistics, compilation, etc. (To appreciate the potential complexity of provided services, cf. the discussion of the National Software Works in [NSW 76].)

Easy access to services in a networking environment requires that one be able to: i) freely access different hosts containing specific service components; ii) simultaneously access multiple hosts; iii) execute jobs on one or several hosts; and iv) manipulate files on a network wide basis. Service construction clearly requires service access and, additionally, requires access to high level nonprocedural mechanisms supporting the use of subnetwork communications and aid in access to different systems. System implementers provide these nonprocedural mechanisms. Their provision, as would be expected, proves difficult because of the significant amount of detailed, system-specific knowledge required for their effective construction.

These considerations coupled with the underlying XNOS/XNOS-II communications characteristics (namely, over the relatively low bandwidth, high delay ARPANET) led to the decision that the system should only support transaction-oriented processing.

This design choice is made both to reduce interhost traffic to avoid possible implementation difficulties. This latter point needs further explanation. A NOS presents, to its user, the appearance of a state-of-the-art timesharing operating system. Therefore, it is natural to expect uniformity of NOS services. However, the provision of common versions of system programs such as editors is very difficult (until such time as they become standardized), since the organizations providing them may be unwilling to incorporate common interfaces, and these system programs may well be (and in fact, generally are) different substantially in many detailed ways. Therefore, we conclude that the alternative of offloading common system-program level support onto local machines presently is far more promising and attractive. This alternative is clearly feasible given the XNOS environment and has the added advantage of cleanly separating the service responsibilities of XNOS implementation from those of the service-providing hosts.

2.3 XNOS/XNOS-II Functionality

Viewed from the perspective given above, the functional objective of XNOS/XNOS-II is to support and unify access to the transaction processing services existing on local hosts. Thus, XNOS/XNOS-II must provide four essential functions [NSW 76], [Kimbleton 80]:

i) global access control and system management,
ii) network job execution,
iii) network data support, and
iv) user interface.

We now address these four NOS functions in turn, describing and comparing the design solutions for XNOS and XNOS-II.

i) Global Access Control and System Management--This includes such functions as logging in and out, network-wide accounting, matching global resource requirements to local resource availability, and advising users of the the existence and accessibility requirements of services.

Since apart from the logging function, neither versions of the system contain any of these features, we shall discuss only the logging function here. Firstly, we note that XNOS-I does not implement any global access control, whereas XNOS-II maintains a network level login/password mechanism such that the user-provided information is given in a standard format, independent of the operating system being accessed.

Secondly, these two versions also differ in their accessing of local host resources. XNOS-I operates in one of two possible modes. Users may interact with local hosts entirely on their own, that is, they must establish accounts with individual hosts, make and break contact with the various local o/s's and conform with the different resource naming conventions of these systems [Fitzgerald 78a]. Alternatively, when the XNOS-I nwds is used [Fitzgerald 78b], users are not given any adminstrative control over local host resources. XNOS-I 'owns' all files resident on the individual systems and maintains control over them. Thus, in this mode, the relationship between the XNOS-I user and local files is much the same as that between a timesharing user and the physical devices of the system.

In contrast, XNOS-II supports diverse local host environments. The specific environment is made selectable on a per user basis. This is achieved by maintaining user-related, local host information such as login names and passwords in a user profile (in encrypted form). This information is used by XNOS-II to gain access to the desired accounts on individual hosts. (After this initial step, there is still the question of how local resources such as files are allowed to be handled by the network user. We shall defer discussions of this topic to item (iii).)

ii) Network Job Execution--This enables users to initiate and monitor the execution of a specified program (or a collection of programs, making up a sequence of job steps), with automatic migration of the output (as appropriate) so that different job steps running on different hosts will be able to access the required data as needed.

Both XNOS-I and XNOS-II provide this capability, though in somewhat different forms. The difference came about because rra is not an integral part of XNOS-I's common command language. Therefore, for XNOS-I, the execution of jobs involving structured files on dissimilar hosts is a two step process: the data must be moved by the user before the job execution can be initiated. XNOS-II, by incorporating record structure description [Wood 80] into its nwds and automating its rra invocation, is able to provide a network-wide program execution environment. It supports one step, location- independent data access and a simple, unified transaction processing capability.

iii) Network Data Support--The system maintains a network-wide file system with access protection to facilitate controlled sharing. It also provides mechanisms for transmitting structured data between dissimilar hosts in a meaning-preserving way.

The network data support machinery for XNOS-I is deficient in two respects: the system makes no provision for files consisting of structured records nor does it implement any privacy or security measures for file accessing. The first limitation results in complicated job execution procedures as discussed above. The second shortcoming results in inadequate support for controlled sharing of network-wide data. Both of these limitations are alleviated in XNOS-II. The former by integrating record structure descriptions into nwds and the latter by associating access protection codes with each network file. The structured record aspect has already been described in item (ii) and details of the protection mechanism will be given later in Section 3.1.

iv) User interface--This provides for handling i/o with the user's terminal and ensures that functionally similar software on different hosts can be invoked in a single, common sequence of commands.

Both versions were able to achieve a very high level of homogeniety of individual host command languages. (This is not surprising, since it is the major goal of XNOS/XNOS-II() But XNOS-II provides additional functions to allow users to maintain run time control over command (or job) execution--i.e., the ability to abort an ongoing process by pressing the 'rubout' key on the user's terminal. A second XNOS-II enhancement is the availability of online ('help') information about the system. Other user services such as a UNIX interface from within XNOS and status information, were common to both versions. (We note that although these are not provided within the XNOS-I system itself, they are available to XNOS-I users since the underlying NAM supports such features [Rosenthal 78].)

This concludes our discussion of the functionalities of XNOS/XNOS-II. In the next section we shall rationalize the above choices by presenting the XNOS system structure.

3. XNOS-II SYSTEM STRUCTURE

This section is organized according to the major categories of XNOS management tasks, that is, the (network-wide) file system, jobs, and access control. Section 3.1 introduces the overall organization. Sec. 3.2 then presents the XNOS-II command language, arranged in the above functional order. Section 3.3 describes the system internals together with the design considerations that resulted in this particular organization.

3.1 Network-Wide System Management

Considerations of the goals and user requirements for XNOS (Secs. 2.1 and 2.2) led us to focus our design effort on three major NOS management functions: network-wide access control, network-wide file system and network-wide job execution environment. This division is derived from the conventional view of transaction processing activity as accessing the system, executing programs using data stored as files and producing output data files. Although to accomodate more general types of usage, it is desirable to include other forms of input/output in addition to files (e.g., devices), the detailed nature of the host specificity of most device handlers persuaded us against it (*).

(*) When such capabilities are needed, we use the individual host system (e.g., o/s or database management system) facility to 'convert' all i/o to that for files [Wang 80b]. This approach should be contrasted to that of National Software Works [NSW 76] whose file package bypasses the individual host's file system and implements its own physical i/o mechanism, and also to RSEXEC [Thomas 73] which provides a "device binding" facility.

We now return to the first design issue: network-wide access control. At the global system level, XNOS-II uses the conventional password mechanism for controlling unauthorized access to its resources. At the individual system level, XNOS-II manages local resources only in the role as an agent for network users, i.e., the accounts that XNOS-II accesses are the user's own accounts on the remote host. This role (or rather, the lack thereof) of XNOS-II in terms of administrative control over individual host resources is consistent with the goals stated in Section 2.1. We want XNOS to be a mediator between the user and the existing resources (accounts, files and programs) on the various network hosts, rather than a manager or administrator of these resources. This decision minimizes the disturbance of other users and other modes of using these resources.

The second issue, network-wide file system, is resolved in much the same manner. The XNOS-II manages remote files on behalf of the network user, but the files are still owned by the users. Consequently, they are not protected from non-XNOS-II activities and interference between these two types of usage is possible.

In addition to providing a unified means for accessing files (i.e., a common file naming convention and location independent accessing), XNOS-II allows network-wide, selective sharing of files. The approach is relatively standard: the nwds maintains, for each file, six bits of access control information together with an owner name. The protection bits are used to specify permission to read, to write and to execute the file for the owner himself and for all others (*).

As indicated in Sec. 2.3, XNOS-II supports a network-wide job execution environment which masks the boundaries between network hosts. This is the most useful and the most essential function for NOSs aimed at transaction processing applications. Systems supporting this kind of environment require heavy communication and processing overhead, though. The reason is that it involves the transporting of data at the record level and the interpretation and processing of each data record. But given the NOS goals and the heterogeneous nature of the constituent network hosts, this demand on system resources can hardly be avoided.

However, we did attempt in XNOS-II to limit the amount of processing overhead. We chose to implement the rje environment for high-level programming languages only. Data transfer between dissimilar hosts preserves just enough meaning to satisfy the needs of (high-level language) programs but not more specialized (and more sophisticated)

(*) Like most timesharing systems [Thompson 75], XNOS-II has the 'super user' loophole incorporated into its protection system.

data access needs, such as database management applications [Date 77]. Thus, we are concerned only with the "meaning" of data at the level that is independent of its physical representation. Differences at the logical representation level (e.g., programs might choose a different unit system from that actually used in the data file [Wood 80]) are not accounted for by the data handling (i.e., the XNOS-II rra) subsystem. This is the responsibility of the users of XNOS-II, that is, the application programs . (This is exactly the same level of program support that a conventional o/s provides. The readers are referred to other work by Wang [Wang 80b] for a detailed discussion of these data representation and translation/transformation issues.) This design choice allows significant reduction of the amount of data conversion during interhost file movement by eliminating data transformations.

3.2 The XNOS-II Command Language

Up to this point we have been concerned with the general aspects of XNOS-II. We shall now describe the system in more concrete terms by presenting its command language.

XNOS-II commands divide into four functional categories and are described under separate headings below. Each command consists of two basic elements: the command name and a list of arguments on which the command operates. The arguments can be either file names or flags for setting particular command options. File names can be specified in one of two ways: either using the nwds name which are strings of 16 or less characters (without any '¶'s), or directly in terms of the remote file name and remote host name combination: filename@hostname.

The format of each command description is as follows: the command name, underlined, is given first followed by the usage line, which summarizes how to use the command. In the usage line, the following conventions apply.

1. Underlined words are considered as literals.
2. Square brackets around an argument indicate that it is optional.
3. Argument names beginning with a hyphen, such as -a, indicate a flag argument.

(1) ACCESSING THE SYSTEM

login	used to gain access to XNOS-II.
Usage:	login [Name] 'Name' is the name of the user. This command will prompt for passwords.
logout or exit	terminates an XNOS-II session.
Usage:	logout, exit
password	changes login password.
Usage:	password [name] The optional name is a system login name. If no name is given the current login identity is assumed. The user is prompted for the current password associated with this name, and then for the one that is to replace it.

(2) FILE SYSTEM COMMANDS

The first four commands manipulate files (or 'segments' in Multics [Organick 72] terminology). The rest are directory manipulation commands.

append	concatenates one file to the end of another.
Usage:	append [-a] [-x] [-i] [file1] [file2] If file1 resides on a different machine from file2, it is moved to file2's host and the appending is performed. The -a flag indicates a stream of ASCII bytes (this is the default). The '-x' indicates structured records and '-i' is used for a stream of bits (no mapping is to take place).
compare	compares two files.
Usage:	compare [file1] [file2] The two files are compared. The byte and the line number at which they differ are output. (This command only applies to ASCII files.)
copy	makes a copy of a file.
Usage:	copy [-a] [-x] [-i] [file1] [file2] file1 is copied into file2. If file2 already exists, it is lost. The meaning of the flags (-a, -x, -i) is the same as for append. '-a' is again the default.
type	writes out the file on tty.
Usage:	type [file] Type only works for ASCII files. If the specified file is not ASCII, unpredictable (file location dependent) results may occur.
create	creates a directory entry for a file.
Usage:	create [file] This command makes an nwds entry for a file already existing on some network host. It prompts the user for certain relevant information concerning the file (remote filename, type, etc.).
delete	removes a file.
Usage:	delete [file] Delete removes the file entry from the nwds. The file still exists and is still known to the host on which the file is located. A

deleted file can be made accessible (to XNOS-II users) again by 'undeleting' it (see below).

erase — removes a file (irrevocably).

Usage: erase [filel
This command completely removes the named file. Once erase is issued the file can not be recovered. An automatic erase is invoked on all 'undeleted' files (see below) when the user logs out.

list — displays file names and file information from the user's nwds or remote host.

Usage: list [-h] [-d] [-u] [-a] [-o] [filel [user]
List displays owner, status (deleted or undeleted), type (ASCII, image or structured), file names (both nwds and remote), protection codes and location information (remote host and owner names). The options -d, -u and -a produce listings of deleted, undeleted or all files, respectively. The option -o produces a listing of other user's files, specified by [user]. The option -h produces a remote host listing of the named file.

rename — changes the name of a file.

Usage: rename [filel] [file2]
Renaming of source to target files residing on different hosts is not allowed. The user may copy the source file to another file on the destination host, then delete the original one.

protect — changes the world protection code of a file.

Usage: protect [filel [world-code]
The protection codes are:

r	read
w	write and
e	execute.

Only the owner of a file (or the super-user) may change its protection code.

undelete — returns any previously deleted files.

Usage: undelete [filel
This command restores a file in the nwds and makes it accessible again.

(3) DISTRIBUTED JOB EXECUTION

run	initiates the execution of a program.
Usage:	run [program] The system queries the user about the files involved, and arranges for the migration of input/output files (by calling the rra service) before/after the actual job execution takes place.

(4) network WITH THE SYSTEM

help	displays help information on various XNOS-II commands.
Usage:	help [command-name] The optional argument (command-name) instructs the system that the user only wants to receive information concerning the specific command.
status	displays current connection information on tty.
Usage:	status

In addition to the above commands, XNOS-II provides a UNIX system interface. Any UNIX command may be specified by prefixing it with a herald '!' (exclamation point). The only exceptions are those commands interpreted directly by the UNIX shell, such as 'login', 'logout' and 'chdir' [Thompson 75]. Another feature is that the user may choose to work directly on the remote system by invoking the 'goremote' command of XNOS-II. This causes the system to enter store-and-forward mode, where all user inputs are sent line-at-a-time to the remote connection and remote responses are sent back to the user's terminal. Typing the command '$bye' returns the user back to XNOS-II.

3.3 System Internals

XNOS-II is designed to operate on geographically dispersed, heterogeneous networks such as the ARPANET . This implementation environment imposes several constraints. First, interhost communication is slow, failure-prone and expensive, and therefore should be used as infrequently as possible. Second, the hosts are operationally and administratively autonomous, and this autonomy must be preserved. Third, the hosts are highly heterogeneous, both in their underlying machine architecture and in their operating systems. This heterogeneity must be accounted for and made invisible to the users.

These constraints established our attitude toward designing XNOS-II: although the system appears as a resource manager to the network user, it should only act as an agent between the user and the remote computing resources. That is, its resource management role is limited within the network level. All managerial and administrative decisions concerning remote resources are made solely by the remote system (at the request of XNOS-II). This basic view has implications on the two most important aspects of the XNOS-II architecture: its file structure (the nwds) and the system model (the organization and partitioning of functions into the various processes making up the system).

XNOS-II implements a fairly simplistic single-level file directory. The information contained within this directory is of three types: i) network-level file management information such as network-wide file name, owner, and protection code, ii) information needed for accessing the file such as its location (host name) and remote owner name, owner project name, and full (remote) path name, and iii) information needed for interhost file movement, that is, data type specification for continuous files and logical record structure descriptions [Wood 80] for structured files.

The only function the file system performs is that of gaining access to the actual file (on behalf of either the network user or a remote job). No file reassignments or movements were done to optimize network-wide resource utilization and therefore no statistics concerning file usage were kept. This is not a defect of the system, since all files are owned by its remote users and global data movements with adverse effects on existing progams is to be avoided.

Another consequence of this design constraint is that we can not take advantage of the distributed nature of XNOS-II to enhance its reliability or robustness. Load balancing is also entirely dependent upon the users themselves, since no file replication, partitioning and placement is done by the system.

XNOS-II is designed only to support transaction processing. Thus, the decision to implement all data movement operations on a file-by-file basis seems sound. The system has one serious weakness in that there is no concurrency control mechanism whatsoever. Thus, two independent "add 2 to variable x"s may indeed result in an increment of only 2 to the variable x [Lampson 80]. To remedy this, some form of locking must be added to the system. This is left for future investigation.

Now let us turn to the XNOS-II system model. XNOS-II is organized into a collection of independent, concurrent, cooperating processes. These processes are of two types: user or server. The user process is the NIM resident portion of the system. It performs the functions of user catering: connecting users to XNOS, accepting and interpreting inputs, manipulating the nwds as instructed, and translating users' requests for remote resources into commands acceptable to the individual hosts. The server process, resident on the remote host, performs the function of interfacing to the remote o/s or file system: initiating the execution of programs and retrieving and delivering data files from the remote system to the XNOS user station.

To simplify the implementation of the system, we chose <u>not</u> to construct our own servers. Rather, we either use the individual host's CLI or its ARPA ftp server for this role. This alleviated some (but not all) of the terminal emulation problem of XNOS-I [Kimbleton 80]. This is because CLIs are no longer needed to handle the most difficult (and most awkward for a CLI) file operation: interhost file transferring. To solve the complete terminal emulation problem, we need to implement server-XNOS's for each of the participating hosts. This should not pose any significant problem, provided a network interprocess communication facility of sufficient power such as MSG [MSG 78] is available. This is left for future work.

To summarize, XNOS-II has a single-level file directory structure. The file system only performs network-level file management and delegates all other administrative tasks (e.g. physical placement of files) to the remote o/s on which the file is located. XNOS-II's implementation is based upon a mixed single and distributed agent model [Thomas 78]. The basic reason for this choice is reduced implementation effort through building as much as possible on existing host software. As has been hinted at before (Sec. 1), this basic structure has advantages as well as disadvantages. This is the topic of the next section.

4. XNOS-II AS A DISTRIBUTED OPERATING SYSTEM

As a distributed system, XNOS-II exhibits the following characteristics. (Enslow's framework for distributed processing systems [Enslow 78] is used for our discussion below.)

The system is built on the hardware base of multiple, interconnected computers. Thus, it is highly decentralized in terms of hardware structure. Its control organization has two separate aspects: network-level (i.e., outside the scope of individual hosts) resources such as network-wide files are managed by a single fixed (both physically and conceptually) entity--the XNOS-II user station. Decisions and actions involved in actual remote resource utilization, on the other hand, are taken by the individual hosts (at the request of XNOS). (To paraphrase Enslow [Enslow 80], we might call XNOS-II a Fully Networked Operating System, as the system plays no role whatsoever in remote resource management.)

The characterization of XNOS-II along the dimension of database organization follows the same pattern as for control organization. The two components of the database--files and directories--are managed entirely differently. The files are distributed onto different hosts, with separate local directories for individual hosts to access and manipulate them. These local directories are external to XNOS-II. The only database for which XNOS-II is responsible is the nwds and the (network) access control list. These are located and managed centrally by the system. An illustration of the above three aspects of XNOS-II is given in Figure 1.

As a distributed operating system, XNOS-II's principal strength is that it provides very effective network transaction processing environments on geographically dispersed networks. Another attraction of the system is that the incorporation of new hosts requires very little additional effort. This is the main reason behind our implementation decision to use existing CLIs and ftp-servers instead of building our own servers.

The weaknesses of the system mostly derive from the fact that it is a <u>centralized</u>, <u>network</u>, and not a <u>distributed</u> operating system. These are:

- users must make global resource management decisions,
- interferences between XNOS and non-XNOS activities are possible,
- because of its centralized implementation, XNOS-II is vulnerable to NIM failures,
- because files are not replicated, XNOS-II is not resilient to failures of constituent hosts, and
- update anomolies are possible for shared files, since there is no concurrency control mechanism.

(This last aspect is an implementation defect rather than a fundamental system limitation.)

5. CONCLUDING REMARKS

The above sections have described the basic structure of the XNOS-II system, the design trade-offs made, the advantages and disadvantages of the particular design choices, and the current implementation status.

The decision to build a centralized, network operating system simplified many of the implementation issues. We did not need complicated mechanisms for coordinating the various parts of, say, a distributed file system. The existing individual operating systems take care of the actual operations of executing jobs and manipulating and managing files. The transaction processing environment avoided many of the performance bottlenecks. Remote data accesses and updates are batched together into one single operation (per file) for each transaction. This allows efficient use of the underlying ARPA communications subnet. It also eliminates the need for encapsulation [Thomas 73], since information concerning data movements can be obtained beforehand through interactive dialog with the user.

The choice of UNIX as the supporting NIM operating system also facilitates the XNOS-II implementation, the greatest advantages being the ease of putting together independently written routines and the availability of a variety of programming tools including the high-level system implementation language C [Kernighan 1978]. The lack of asynchronous i/o and the restrictive nature of the UNIX interprocess communication mechanism did complicate XNOS-II programs. However, these are solvable (although not in as satisfactory a way as one would like) problems [Balocca 78].

The total XNOS system design (and redesign) gave us an opportunity to compare the various ways of partitioning and assigning (or delegating) control responsibilities for network as well as remote resources. This resulted in the construction of a centralized, "fully networked" operating system. The central system functionalities are offloaded from constituent network hosts onto a special purpose computer, the NIM. Instead of either having users deal with remote hosts individually (cf. the first mode of XNOS-I operation [Fitzgerald 78a]), or having one set of remote resources (owned and managed by the NOS, cf. the second mode of XNOS-I operation [Fitzgerald 78b]), XNOS-II supports a much more flexible NOS/individual host relationship: it allows multiple remote environments, selectable on a per user basis.

So far, our experience with the system indicates that the structure of XNOS-II does provide a good base for building reliable and effective network-wide processing environments for the support of transaction applications. However, improvements and enhancements to the system are still needed in many respects (see Secs. 2.3 and 3.3). We hope that future work will provide more concrete evidence of the soundness of the XNOS-II approach and also will bring about the necessary improvements to the system.

REFERENCES

[Abrams 74] Abrams, M. D. Measuring Service Delivered in Interactive Computing. Proc. Second Jerusalem Conf. on Inf. Tech., Aug. 1974, pp.7474-754.

[Balocca 78] Balocca, R. Networking and the Process Structure of Unix: a Case Study. Proc. COMPCON 78 Fall - Computer Communications Networks, Sept. 1978, pp. 306 - 311.

[Chesson 75] Chesson, G. L. The Network UNIX System. Proc. Fifth Symp. on Operating System Principles, Univ. of Texas, Austin, Tex., Nov. 1975, pp.60-66.

[Date 77] Date, C. J. "An Introduction to Database Systems", 2nd Ed., Addison-Wesley, Reading, Mass., 1977.

[Enslow 78] Enslow, P. H., Jr. What is a "Distributed" Data Processing System? Computer, Vol.11 (1978), 13 - 21.

[Enslow 80] Enslow, P. H., Jr. FDPS (Fully Distributed Processing System) Research at Georgia Tech., Talk presented at the Rome Air Force Development Center Technology Exchange Meeting, Rome, N. Y., May 13 -15, 1980.

[Feinler 78] Feinler, E. and Postel, J. ARPAnet Protocol Handbook. NIC 7104, Network Information Center, SRI International, Menlo Park, Calif., Jan. 1978.

[Fitzgerald 78a] Fitzgerald, M. L. Common Command Language for File Manipulation and Network Job Execution: An Example. NBS of Standards SP 500-37, National Bureau of Standards, Washington, D. C. 20234, Aug. 1978.

[Fitzgerald 78b] Fitzgerald, M. L. XNOS Integrated Command Language Code Documentation. Internal Report, National Bureau of Standards, Washington, D. C. 20234, Apr. 1978.

[Forsdick 78] Forsdick, H. C. , Shantz, R. E. and Thomas, R. H. Operating Systems for Computer Networks. Computer, Vol. 11 (1978), 48 - 57.

[Frank 72] Frank, H., Kahn, R. E. and Kleinrock, L. Computer Communication network design-Experience with theory and practice. Proc. AFIPS SJCC, Vol.40 (1972), 255 - 270.

[Jensen 80] Jensen, E. D. Local, Global, Distributed, Centralized and Network Operating Systems. To appear in "An Advanced Course on Distributed Systems - Architecture and Implementation", M. Paul and H. J. Siegert, Eds., Springer-Verlag, New York, 1980.

[Kernighan 78] Kernighan, B. W. and Ritchie, D. M. "The C Programming Language", Prentice-Hall, Inc., Englewood Cliffs, N. J., 1978.

[Kimbleton 78] Kimbleton, S. R., Wood, H. M. and Fitzgerald, M. L. Network Operating Systems - An Implementation Approach. Proc. AFIPS NCC, Vol.47 (1978), 773 - 782.

[Kimbleton 80] Kimbleton, S. R. Network Operating Systems: Perspective and High Level Interface Issues. (In preparation). National Bureau of Standards, Washington, D. C. 20234, 1980.

[Lampson 80] Lampson, B. W. Atomic Transactions. To appear in "An Advanced Course on Distributed Systems - Architecture and Implementation", M. Paul and H. J. Siegert, Eds., Springer-Verlag, 1980.

[MSG 78] MSG: The Interprocess Communication Facility for The National Software Works. Bolt Beranek and Newman, Inc. , Aug. 1978.

[NSW 76] National Software Works. Semi-Annual Technical Report. CADD-7603-0411, Mass. Comp. Assoc., Wakefield, Mass., Mar. 1976.

[Organick 72] Organick, E. I. "The Multics system: An examination of its structure", MIT Press, Cambridge, Mass., 1972.

[Ritchie 74] Ritchie, D. M. and Thompson, K. The UNIX Time-Sharing System. Comm. ACM, Vol.17 (1974),365 - 375.

[Ritchie 78] Ritchie, D. M. UNIX Time-Sharing System: A Retrospective. The Bell Sys. Tech. J., Vol.57 (1978), 1947 - 1969.

[Rosenthal 78] Rosenthal, R. M. and Lucas, B. D. The Design and Implementation of the National Bureau of Standards' Network Access Machine (NAM). NBS SP 500-35, National Bureau of Standards, Washington, D. C. , Jun. 1978.

[Thomas 73] Thomas, R. H. JSYS - A TENEX Mechanism for Encapsulation of User Processes. Proc. AFIPS NCC, Vol. 42 (1973), 155 - 163.

[Thomas 78] Thomas, R. H., Shantz, R. E. and Forsdick, H. C. Network Operating Systems. Final Tech. Rep. RADC-TR-78-117, Rome Air Development Center, Air Force

Systems Command, Griffiss Air Force Base, New York 13441, May 1978.

[Thompson 75] Thompson, K. and Ritchie, D. M. UNIX Programmer's Manual. Bell Laboratories, Inc., May 1975.

[Wang 80a] Wang, P. S.-C. and Shertz, J. L. XNOS-II Reference Manual. (In preparation). National Bureau of Standards, Washington, D. C. 20234, 1980.

[Wang 80b] Wang, P. S.-C. and Kimbleton, S. R. A Data Transfer Protocol for Remote Database Access. Proc. Trends & Applications:1980--Computer Network Protocols, May 1980, pp.70-82.

[Wood 80] Wood, H. M. and Kimbleton, S. R. Remote Record Access: Requirements, Implementation and Analysis. NBS SP 500- (In preparation). National Bureau of Standards, Washington, D. C., 1980.

Blank

V

DATABASE SEMANTIC INTEGRITY FOR A NETWORK DATA MANAGER

Elizabeth N. Fong

Stephen R. Kimbleton

National Bureau of Standards

May 1980

ABSTRACT

Data in a database represents an abstraction or model of a 'real world' application. Effective use of the database requires that the contained data accurately describe the application. The field of semantic integrity is concerned with assuring that data is logically correct. Semantic integrity promises to be of even greater importance in the context of networked databases since local database management is likely to want strong assurances that remote, and therefore, presumably less knowledgeable users, will not impact database integrity. This paper: i) categorizes semantic integrity features; ii) identifies an approach to maintaining integrity in accessing multiple, remote, heterogeneous DBMSs; and iii) describes an Experimental Semantic Integrity System (XSIS) now being designed and implemented at the National Bureau of Standards.

INTRODUCTION

The goal of semantic integrity is assuring that data within a database is logically correct. Logical correctness is evaluated by showing that the data represents a valid abstraction or model of a 'real world' application. This is required for effective use of the database.

The starting point in assuring database semantic integrity is a clean (semantically correct, i.e. logically correct) database and the objective is ensuring that subsequent updates also result in a clean database. Since individual updates may involve operations across data structures, may require several statements of the data manipulation language being used, and may involve logically interrelated data, assuring semantic integrity is a difficult problem which currently lacks a complete solution.

This paper has two major objectives. The primary objective is establishing a collection of capabilities appropriate for a semantic integrity system. This assumes an environment providing a uniform mode of access for the network user to multiple, remote heterogeneous DBMSs. The secondary objective is providing a brief survey of the existing semantic integrity literature. Thus, the following two sections explore the existing approaches to semantic integrity. Thereafter, the environment supporting remote database access is described. This is followed by a discussion of the design of an Experimental Semantic Integrity System (XSIS) currently being constructed at the National Bureau of Standards.

WHAT IS SEMANTIC INTEGRITY?

Data within a DBMS can be in error for one of three reasons: incorrect entry, system failure, or conflict with the intended meaning of the data. Incorrect entry occurs when a wrong value gets keyed-in during the data entry process. For example, due to a keypunch error, a person's weight might be entered as 87 rather than 78 kgs.

System errors reflect a variety of failures in design or implementation, including: security violations, hardware failures, and failures in system or DBMS software such as the concurrency update problem.

Conflicts with the intended meaning of data occur through violations of perceived interrelationships among the data within DBMS. For instance, the sum of division capital expenditure budgets, after an update, may no longer equal the organizational capital expenditure budget.

Semantic integrity is concerned with assuring that such violations cannot occur. Since such violations represent a conflict with the understood meaning of data, and since this understood meaning is usually not formally expressed, the goal of achieving semantic integrity has proved elusive.

Because semantic integrity is concerned with the perceived correct meaning of data, it can also provide limited assistance in detecting data entry errors or system failures. For instance, through value-range specification, the semantic integrity system could detect the error in entering a person's weight as 570 kg. rather than 70 kg.

It could not detect the difference between 78 and 87 kgs. Similarly, system failures resulting in major data errors can be detected when the data is examined.

Approaches to Semantic Integrity

There are two primary approaches to ensuring semantic integrity. The assertion based approach permits users or Database Administrators to specify those semantic integrity rules which seem important. This approach facilitates incremental incorporation of semantic integrity within the DBMS environment.

The disadvantages of the assertion based approach are potentially significant and include: i) the inability to determine the completeness of the collection of rules since there is no basic point of reference for their assessment, ii) the possibility that adding a new rule will show that a database previously thought to be semantically correct now suffers from integrity violations, and iii) the need for explicit incorporation of consistency checks on the collection of rules, which might lead to significant overhead.

The major alternative to the assertion based approach depends on a thorough development and application of the concept of type [BRODM 78]. Strong data types are constructed from system data types. Moreover, through exploiting the concepts of aggregation and generalization, type interrelationships can be formally specified. Such specification has the advantage of permitting semantic integrity assertions to be checked statically at compile-time rather than run-time. Additionally, the completeness and consistency of data type specifications can be shown.

The disadvantage of this approach is the explicit requirement for complete specification. For databases of significant size, completion of the specification process requires a major investment of effort. Another drawback is that not all of semantic integrity requirements can be specified using the data type concept because certain validity criteria are "value" rather than "type" oriented. A combined approach may prove best in an operational environment.

<u>Semantic Integrity System</u>

A combined approach to supporting semantic integrity seems appropriate. Implementing such a system requires specification of three major components: i) the rules specifying the semantic integrity constraints which may be either type or value constraints, ii) the process for checking conformance with these rules, and iii) the actions which occur upon detection of an integrity violation.

<u>Rules Specification</u> - The semantic integrity rules or constraints need to be stated prior to data base use. These constraints specify all the required information to be used during rules enforcement time.

Strong data type specifications effectively augment the traditional concept of schema and are stated through denotational or declarative methods at data definition time. Assertion based approaches require a means for specifying the assertion and are evaluated while a request is being processed.

Designing a semantic integrity specification language requires considering two main issues: i) the style of the specification language used by users and database administrators, and ii) the volume of information required by the system to check conformance with the rules.

Rules Enforcement - Evaluating conformance with semantic integrity rules can be thought of as being performed by an abstract observer monitoring database operations. The times at which the observer can observe the database are defined, and there may be times during an update when the observer is precluded from judging whether semantic integrity is being maintained.

Implementing the observer requires considering three major issues: i) the types of tests which the observer may perform, ii) the information required to support these tests, and iii) the cost of performing them. This last item is very important from a practical point of view.

Violation Actions - Detection of a semantic integrity violation requires flagging the error and, probably, rejecting the update. The precise integrity rule(s) which are violated should be identified. Error recovery options include calling a user-specified error routine, reporting an error message to the user, or semi-automated error elimination by the system.

Related Works on Semantic Integrity

Much of the existing semantic integrity literature is concerned with isolated aspects of the global problem - placement of responsibility, data semantics, specification languages, invocation techniques, and supporting system environments. For instance, [FERNE 76], [GRAVR 75], and [MACHC 76] all describe high level semantic integrity specification languages. In [WEBEH 76] semantic integrity is viewed in the context of state transitions; therefore, constraints are expressed upon database operations. Buneman and Morgan [BUNEO 77] developed "alerting" mechanisms for supporting semantic integrity. Several selected semantic integrity systems are reviewed below.

Brodie - Brodie's approach [BRODM 78] views semantic integrity as a (semantic integrity) system rather than user responsibility. A specification language together with a verification methodology is developed. The specification language is based on a denotational approach and the emphasis is on proving consistency and completeness.

Since Brodie's approach is not currently implemented, support system requirements have not been identified. However, the extremely systematic approach which is presented requires complete specification at database design time. Operational use would require investigation of the issues of sizing and flexibility.

McLeod - McLeod's approach [MCLED 76, HAMMM 75] is assertion based, and views semantic integrity as a system responsibility. The semantic integrity system he describes places special emphasis on the design of a constraint specification language for a relational data model. The specification language is based on using a high level, non-procedural language permitting specification of: i) constraints, ii) times at which the constraints are to hold, and iii) actions to be taken on occurrence of a constraint violation.

System R - System R, a relational database management system developed at IBM, provides semantic integrity facilities as part of the SEQUEL language [ESWAK 75]. The approach is based on assertions which can be any SEQUEL predicate evaluating to a Boolean. System R provides a very extensive collection of semantic integrity support features including: tuple and set assertions, state and transition assertions, and immediate and delayed assertions. Semantic integrity enforcement is implemented using system triggers and, if the result of a transaction is proven to violate an assertion, the transaction is rejected and a predefined procedure or failure action is invoked.

INGRES - INGRES is a relational database management system developed at the University of California, Berkeley. Semantic integrity assertions in INGRES have the form of one or more range statements plus an integrity qualification. These assertions, in an intermediate representation, are appended to all user updates. This query modification technique minimizes the requirements for additional software. It also limits the set of semantic integrity features which can be provided.

CODASYL - The CODASYL Data Description Language Committee Journal of Development specification [CODAS 79] provides a user-written procedure invocation mechanism which allows integrity requirements to be programmmed in procedural code by the user (applications programmer). The keyword CHECK with user (applications programmer) specified parameters provides the triggering mechanism when a data item is changed. The CODASYL specification does not provide a semantic integrity support system; rather the burden of specification and enforcement of semantic integrity is placed on application programmers.

The CODASYL specification supports an underlying network data model with stringent structural requirements; therefore, facilities for checking for duplicates, member record insertion conditions and uniqueness of keys are supported.

[MELOR 79] proposes some semantic integrity facilities to be incorporated in a CODASYL-like DBMS. These integrity constraints are meant to enhance those provided by the CODASYL specification and those inherent in the network data model.

SEMANTIC INTEGRITY FEATURE LIST

The above brief survey reveals that currently, there is no unified, practical, comprehensive and complete approach to database semantic integrity. Semantic integrity features offered within a DBMS range from very simple data type checking to complex assertion processing. We present a gross feature list with the following categorizations:

Strong Data Type Constraints

Semantic integr'ty constraints based on an extension of the data type concept can be specified at data definition time. Such constraints are related to the concept of strong typing developed for programming languages.

The concept of a strong data type arose from developments in abstract data types [LISKB 74; GUTTG 77] which can be informally described as special purpose entities with constrained usage properties. Constraints typically include both the operations which may be performed on a given data element as well as the collection of other data elements which may be involved in binary operations.

Some examples of strong data type constraints are described as follows:

1. Value constraints - specify the range of acceptable values, establish whether a value must be specified, and define whether a data value must be unique. For instance, data values may be required to lie within certain bounds (e.g., age between 1 to 100). Data values may also be constrained to an enumerated set (e.g., colors are red, green, blue). Data values may be specified to be essential or non-missing in which case missing values are considered to be

semantically incorrect (e.g., data element EMPNO must not be missing). Data values may be required to be unique, (e.g., EMPNO must be unique).

2. Extended format constraints - permit specification of a data format pattern composed of primitive types, such as character, integer or real. For example, the first character of a supplier number may be required to be the letter S.

3. Domain compatibility - support assurance that cross domain operations, e.g. binary operators, are applied to compatible units and prohibits such operations on incompatible domains. For example, automatic invocation of scaling factors is required when some weights are expressed in kilograms and others in pounds. Some automatic techniques for performing such conversions are given in [KARRM 78]. Invalid comparisons, e.g. comparing weight to time should be flagged as a constraint violation.

4. Legal operation constraints - limit the operations which can be performed on a given domain to those judged semantically correct. Thus, a set of legal operations may be associated with a data item. Attempting to perform any other operation will result in a semantic integrity violation, e.g., SSN * 2 or NAME > "123" will be detected as semantically incorrect.

Constrained Data Dependency Assertions

Integrity constraints may involve more than an individual data element. The constraint expresses a condition or predicate on the subset of the database to which the constraint applies. Such constraints across data elements can be further categorized:

1. Data Grouping Constraints - reflecting logical relationships among data elements. For example, if the data is represented in tabular form, there can be a constraint that is dependent upon the other values in the same row, (row constraints), or column (column constraints), or table (table constraints), or collection of tables (inter-table constraints). The constraint: salary of employee must be less than salary of manager, would be a table constraint if salary information and salary of manager appeared in the same table, and an

inter-table constraint if this information appeared in two different tables.

2. Aggregate Function Constraints - based upon built-in aggregate functions such as average, minimum or maximum. For example, average salary may be constrained to be less than $15,000.

3. Data Model Constraints - based upon the particular characteristics of a data model such as relational, hierarchical or network.

Transition Constraints

Transition constraints specify relationships between old and new states of the database; two basic types of transition constraints are:

1. Old/New transition constraint - During an update operation, the "old" value must be in some given relation to the "new" value. For example, new salary must be greater than old salary.

2. Nonexistence/existence transition constraint - The insertion operation involves a nonexistence to existence transition, while a deletion involves an existence to nonexistence transition. For example, deletion of an account number may require that the balance be zero.

User-Controlled Enforcement

Using one of the constraints specified above requires establishing when the constraint is to be invoked. User controlled enforcement permits the user to state WHEN to enforce the integrity constraints:

1. Deferred/immediate enforcement - for transactions requiring more than one data management request, the user may permit suspension of integrity constraints until all requests have been issued. The user is responsible for specifying deferred enforcement and the system must be able to back-out the

transaction when deferred enforcement results in a constraint violation.

2. ON/OFF enforcement - permits the user to switch integrity check ON or OFF depending upon the level of integrity needed for the application. It is useful since some integrity checking is costly.

Integrity Specification and Maintenance Facilities

Some integrity checking, such as simple data type checking, can be automatically incorporated within a database system without the need for user specification. More complex kinds of assertions need to be stated, maintained, and invoked at the appropriate time. Some semantic integrity system facilities might be:

1. User-written Application Program Interface - The DBMS may not to provide a centralized semantic integrity system but provide, instead, interface mechanisms so that users can code their own integrity requirements as an application program. This application program interface is usually available in most of the commercially available DBMS.

2. Integrity Specification Language - The semantic integrity system permits users to specify integrity constraints in a high-level language. This language is compiled into procedures which are triggered for the enforcement of the constraints.

3. Integrity Constraint Maintenance - The semantic integrity system permits users to read, modify, create and destroy integrity constraint assertions. Global constraint consistency should be checked when new constraints are added or existing constraints are modified.

Feature Comparisons

Not all of the features identified above are offered by each of the reviewed systems. Moreover, substantial differences exist in the implementation of features common to two or more systems. Brodie's approach has not yet been implemented. McLeod's has been partially implemented. System R has not been released as a product. INGRES is available and is being used at several sites. CODASYL specifications are still in the process of enhancement. Implemented versions of CODASYL-like systems usually include the CHECK clause plus interface mechanisms for user-written procedures so that necessary integrity features may be coded by the user via application programs.

The semantic integrity feature list contained in Figure 1 together with the various enforcement approaches provides a basis for the design of a prototype Experimental Semantic Integrity System.

XSIS SUPPORT ENVIRONMENT

A prototype Experimental Semantic Integrity System (XSIS) is currently under construction at the National Bureau of Standards. XSIS is designed to operate as part of a distributed, networked environment supporting access to multiple, remote, heterogeneous DBMSs as provided by an Experimental Network Data Manager (XNDM) [KIMBS 79]. Since this paper explores the issues in defining, implementing and maintaining XSIS based on XNDM support, it is important to have an understanding of XNDM's basic structure and of its relationships with network accessible DBMSs.

Figure 1 - Semantic Integrity Feature List

FEATURES
STRONG DATA TYPE CONSTRAINTS
1. VALUE CONSTRAINTS
2. EXTENDED FORMAT CONSTRAINTS
3. DOMAIN COMPATIBILITY
4. LEGAL OPERATION CONSTRAINTS
CONSTRAINED DATA DEPENDENCY ASSERTIONS
1. SINGLE DATA
2. GROUPS OF DATA
3. AGGREGATE FUNCTIONS
4. DATA MODEL
TRANSITION CONSTRAINTS
1. OLD/NEW
2. NONEXISTENCE/EXISTENCE
USER-CONTROLLED ENFORCEMENT
1. DEFERRED/IMMEDIATE
2. ON/OFF
SPECIFICATION & MAINTENANCE FACILITIES
1. USER-WRITTEN PROGRAMS
2. INTEGRITY SPECIFICATION LANGUAGE
3. CONSTRAINTS MAINTENANCE

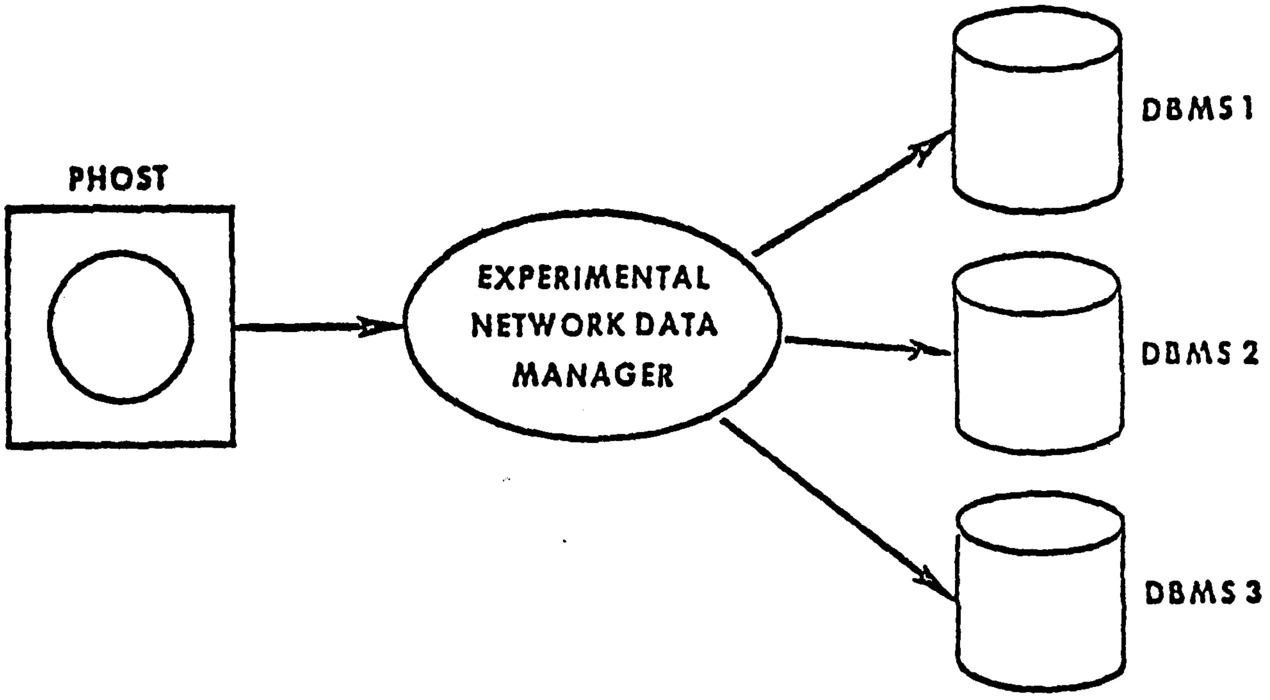
Experimental Network Data Manager (XNDM)

XNDM is implemented in the C language on a PDP-11/45 attached to the Arpanet and running the UNIX operating system. It's basic structure is illustrated in Figure 2. XNDM provides a uniform, table based virtual view of data maintained by independent, network accessible DBMSs. Server modules exist on target systems to provide local support. It differs from a distributed database in that: i) the view of data provide^d to the network user is virtual, ii) the data manipulation language (DML) provided to the network user differs from that actually employed by the target DBMSs, and iii) different systems can use different DBMSs, data models, and DMLs.

XNDM Data Structures - XNDM data structures are assumed to be constructed by a committee of Data Base Administrators who, collectively, identify the data whose access is to be supported and specify the appropriate access paths. Although the virtual view of data presented by these structures is relational, i.e. table based, the data models used by the target systems can be relational, hierarchical, or CODASYL.

XNDM Data Language - The data language for XNDM is termed the Experimental Network Data Language (XNDL). It consists of three major components: a data definition language defining the tables and their data attributes, a data control language providing non-discretionary access controls and semantic integrity specifications, and a data

Figure 2 - XNDM Overview



manipulation language for issuing queries and updates. The DML is based on a subset of SEQUEL (redundant predicates, sort operations, and a programming language interface are excluded) which has been extended to provide primitives appropriate to specifying and controlling concurrent access to multiple databases. A more complete specification of XNDL is contained in [KIMBS 79].

Interfacing to the Target Systems - Using XNDM requires the DML statements (XNDL queries or updates) to be translated to the DML employed by the target system(s). Since this translation process is dependent upon both the preestablished source data structures as well as the locally determined target data structures, it proves substantially harder than that required to support database front ends. Substantial work has however been done on XNDM query translation [KIMBS 79], [WANGP 80].

Currently, XNDM does not support updates. A major technical problem in their support is related to the recognized problem of updating views. A general solution to this problem appears difficult if not impossible [DAYAU 78] because of hidden data. Although this is valid for a general, and hence arbitrary view, it may be resolvable in the context of XNDM applications by suitably defining the data structures presented to the user. Such definition would require classification of DBMS data into transitively closed, independent groups coupled with a requirement that a user be a^le to update all data elements of the group if the user can update any data element of the group. Since remote users often have specialized interests, this approach may be commensurate with implicit organizational requirements.

Deferral of XNDM update capabilities reflects the desire to complete implementation of query support since a thorough understanding of this problem is required to provide the basic addressability information required to support updates. In view of the strong interest which has been expressed to the authors about the impact of the network user on the quality of data maintained by a DBMS, it seemed appropriate to establish the types of guarantees which can be provided together with the problems implicit in their support. This is the objective of this paper.

Need for Semantic Integrity in XNDM - The preceding suggests that local acceptance of a remote updating capability may well depend upon the extent to which suitable correctness guarantees can be provided. Two basic support mechanisms for providing such correctness can be developed. The first is an appropriate collection of discretionary access controls for assuring that individual remote users can only access data appropriate to their access rights. The second is a semantic integrity support capability. Issues related to the first have been discussed in [WOODH 79]; the remainder of this paper is concerned with the second topic.

Semantic Integrity in the Networking Environment

Semantic integrity in the XNDM environment differs in several essential ways from that usually associated with centralized DBMSs.

1. Global vs. local integrity controls - reflecting the possibility of a conflict between globally established constraints and those established by local DBMS management.

2. Conflicting Assertions - constraints appropriate to different (local) DBMSs may be in conflict.

3. Partial data problem - local semantic integrity assertions may involve data not accessible to the network user. Thus, it may prove impossible to provide global checking which is complete from the local point of view.

XSIS DESIGN AND IMPEMENTATION ISSUES

XSIS is also being implemented in C on the same system on which XNDM is being implemented. Its design and implementation are intended to support exploration of both the assertion based and strong data type approaches to semantic integrity. Because of the preliminary nature of XSIS work, the following comments are provisional; based upon accrued experience, the basic XSIS design goals may be substantially modified in the future.

XSIS Design

The major goal of XSIS is to provide a filter for checking remote database operations expressed via the XNDL. XSIS views semantic integrity maintenance as a system rather than user responsibility.

The offloading of XSIS from both target systems and their DBMSs permits a flexible, iterative approach to its design, implementation and modification. The design of XSIS is guided by several principles:

1. Offloading of semantic integrity enforcement - the burden of enforcing semantic integrity constraints on access by the network user should be offloaded, as much as possible, onto XSIS. This reflects the perception that local management may prove unwilling to incur an extra processing burden just to support remote users.

2. Emphasis on global checking - because of the belief that remote users are less knowledgable users, and the desire to minimize local system overhead in supporting semantic integrity.

3. Modular design and implementation - to permit an iterative design, implementation, operation and modification cycle without impacting local systems or other XNDM components.

XSIS consists of two major components: One component supports constraint specification and maintenance. The constraint specification processor accepts a constraint specification and stores an intermediate representation in the semantic integrity tables which are linked to the XNDM data dictionary.

The second component supports constraint evaluation and enforcement. It is activated when a user issues a request in XNDL. Enforcement decomposes into those checks which can be performed independently of the target DBMSs, and those requiring retrieval of data from local databases. The former are primarily type checks while the latter test whether the appropriate assertions are satisfied.

The XSIS processing sequence consists of: i) receipt of a parsed tree representation of an XNDL statement by the XNDM translator, ii) performance of type constraint tests, iii) retrieval of dependent data required to process run time tests, iv) performance of run time tests, and v) return of a condition code indicating whether a semantic integrity violation was detected and, if so, its type.

XSIS Strong Type Constraints - XSIS supports strong data types whose specification includes:

1. Data name.

2. Data format - A description of the format of the data type as composed from primitive types such as character, integer, or real.

3. Legal operators - A list of permissible operators for a given data element, e.g. arithmetic, relational.

4. Compatible domain names - A list of data names which can be involved in binary operations with the given data element.

5. Value restriction - Assertions upon value such as legal range or permitted set of values.

This initial collection of capabilities is significantly less encompassing than those reported in [BRODM 78] but does provide a reasonable range of functionality. Moreover, as experience is gained in their utility, additional extensions can be implemented because of the modular nature of XSIS.

XSIS Assertion Processing - Key issues in supporting XSIS assertion processing are: i) maintenance of the appropriate database of assertions, ii) evaluation of the collection of assertions for consistency, and iii) utilization of these assertions in performing the appropriate run time checks. The first two issues are currently being investigated. The third, given the nature of the networking environment, is affected by data movement requirements necessary to support run time processing. Such data movement can be reduced by having the local node perform some of the checking. This distributed approach has the obvious disadvantage of requiring local additions and modifications. In an operational environment, however, such modifications may prove highly cost effective.

XSIS assertions may involve interrelated data elements. Since a table based data model is being employed, such assertions can be classified into row, column, intra-table, or inter-table assertions. Evidently, processing of more complex assertions is likely to require retrieval of greater amounts of data from the supported DBMSs.

Consistency constraints can be further classified as: change rules, insertion rules, and deletion rules. Initially, only change rules are being implemented.

The distributed environment provided by XNDM imposes a bandwidth limitation on communications between the requesting process and the target DBMSs. As a result, it is highly desirable that such interactions have a transaction oriented flavor as observed in [GRAYJ 78]. In turn, this requires capability for user specification of WHEN semantic integrity rules are to be enforced. Since XSIS is external to the target DBMS, such temporary extensions are easy to effect. However, backing out an extension may prove rather difficult if the target system does not provide an appropriate collection of backout mechanisms.

<u>XSIS System Issues</u> - XSIS semantic integrity specifications are maintained in tables associated with the XNDM data dictionary. Integrity maintenance commands are provided for DISPLAYing, DELETEing, DEFINEing and CHANGEing integrity specifications.

The cost of achieving a high degree of database semantic integrity may be prohibitive [BADAD 79; HAMMM 78]. The distributed nature of semantic integrity enforcement provided by XSIS naturally raises the issue of performance. Consequently, timing and frequency statistics are to be maintained with the integrity specifications for assessing invocation rates and performance penalties. Such information should permit estimating the cost and performance of assuring semantic integrity in the network database environment.

CONCLUDING REMARKS AND IMPLEMENTATION STATUS

The preceding discussion structured a basic approach to providing semantic integrity while supporting uniform access to multiple, remote, heterogeneous DBMSs. The discussion has shown that the implementation framework of XSIS permits a blend of both strong data type and assertion based approaches to semantic integrity. Thus, a flexible vehicle supporting future research in this area has been achieved.

XSIS implementation is currently in the initial phase. Some strong data typing capabilities exist and the implementation of assertion checking has begun. Although the early implementation status precludes reporting operational results, the structure has been described to encourage discussion of this important issue.

ACKNOWLEDGMENT

The authors would like to express their appreciation to L. J. Miller and Pearl S.-C. Wang for their helpful comments on earlier versions of this paper.

REFERENCES

[BADAD 79] Badal,D.Z., "On Efficient Monitoring of Database Assertion in Distributed Databases", Proceedings of 4th Berkeley Conference on Distributed Data Management and Computer Networks, August 1979, pp. 125-137.

[BRODM 78] Brodie, Michael, L. "Specification and Verification of Data Base Semantic Integrity," University of Toronto, Computer System Research Group, Technical Report CSRG-91, April 1978.

[BUNEO 77] Buneman, O. Peter and Howard Lee Morgan, "Implementing Alerting Techniques in Database Systems", The Wharton School, University of Pennsylvania, DDC No. AD-A057319, Feb. 1977.

[CODAS 79] CODASYL Data Description Language Committee, "Data Description Language - Journal of Development, 1978". Available from CODASYL, P.O.Box 1808, Washington DC, 20043, updated thru July 1979.

[DAYAU 78] Dayal, Umeshwar and Philip A. Bernstein, "On the Updatability of Relational Views", proceeding of 4th International Conference on Very Large Data Base, West-Berlin, Germany, Sept 1978, pp. 368-377.

[ESWAK 75] Eswaran, Kapali P. and Donald D. Chamberlin, "Functional Specifications of a Subsystem for Data Base Integrity," proceeding of 2nd International Conference on Very Large Data Base, Farmingham, MA. Sept 1975, pp. 48-68.

[FERNE 76] Fernandez, Eduardo B. and Rita C. Summers, "Integrity Aspects of a Shared Data Base," AFIPS Conference Proceeding, National Computer Conference, Vol. 45, 1976.

[GRAVR 75] Graves, Robert W., "Integrity Control in a Relational Data Description Language," Proceedings of ACM Pacific Regional Conference, 1975, pp. 108-113.

[GRAYJ 78] Gray, James, "Notes on Data Base Operating Systems," in Goos and Harmanis, (eds.) Operating Systems - An Advanced Course, Lecture Notes in Computer Sciences, Vol. 60, pp. 393-481.

[GUTTJ 77] Guttag, J. V., "Abstract Data Types and the Development of Data Structures," Communications ACM 20, 6 (June 1977), pp. 396-404.

[HAMMM 75] Hammer, Michael and Dennis J. McLeod, "Semantic Integrity in a Relational Data Base System," Proceeding of 2nd International Conference on Very Large Data Base, Farmingham, MA. sept 1975, pp. 25-47.

[HAMMM 76] Hammer, Michael and Dennis J. McLeod, "A Framework for Data Base Semantic Integrity," in proceedings of 2nd International Conference on Software Engineering, San Francisco, CA, Oct. 1976.

[HAMMM 78] Hammer, Michael and Sunil K. Sarlin, "Efficient Monitoring of Data Base Assertions", Proceeding of ACM SIGMOD 1978, pp. 159.

[KARRM 78] Karr, Mechael and David B. Loveman III., "Incorporation of Units into Programming Languages" Communications of ACM, Vol. 21, No. 5 (May 1978), pp. 385-391.

[KIMBS 79] Kimbleton, S.R., Pearl Wang and Elizabeth Fong, "XNDM: An Experimental Network Data Manager", Proceedings of the 4th Berkeley Workshop on Distributed Database Management, August 1979, pp. 3-17.

[LISKB 74] Liskov, B.H. and S.N. Zilles, "Programming with Abstract Data Types," Proceedings of ACM SIGPLAN Symposium on Very High Level Language, SIGPLAN Notices 9,4 (April 1974), pp. 50-59.

[MACHC 76] Machgeels, Claude, "A Procedural Language for Expressing Integrity Constraints in the Coexistence Model," in Nijssen, G. M. (ed.) Modelling in Data Base Management Systems, North Holland Publishing Company, 1976.

[MCLED 76] McLeod, Dennis, J., "High Level Expression of Semantic Integrity Specifications in a Relational Data Base Systems," MIT Report MIT/LCS/TR-165, available from DDC AD-A034184, 1976.

[MELOR 79] Melo, Rubens N., "Monitoring Integrity Constraints in a CODASYL-like DBMS", Proceedings of 5th International Conference on Very Large Database, Rio de Janeiro, Brazil, Oct. 1979, pp. 209-218.

[STONM 75] Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification," Proceedings of ACM SIGMOD International Conference on the Management of Data, San Jose, California, May 1985.

[WANGP 80] Wang, P.S.C. "The Design of a Network Virtual Query Language - XNQL. (in preparation).

[WEBEH 76] Weber, Herbert, "A Semantic Model of Integrity Constraints on a Relational Data Base", in Nijssen, G. M. (ed.) Modelling in Data Base Management Systems, North Holland Publishing Company, 1976.

[WOODH 79] Wood, Helen M. and Stephen R. Kimbleton, "Access Control Mechanisms for a Network Operating System", Proceeding of National Computer Conference, June 1979.

VI

ELEMENTS OF A TECHNICAL SPECIFICATION OF AN EXPERIMENTAL SEMANTIC INTEGRITY SYSTEM

Elizabeth N. Fong

National Bureau of Standards

September 1980

ABSTRACT

Database semantic integrity refers to the logical correctness of databases and the assurance that all updates to the databases result in consistent states. In this report, we give elements of a technical specification of an Experimental Semantic Integrity System (XSIS) for a distributed, networking environment. The XSIS should: i) provide a semantic integrity specification language expressed at the global schema level, ii) check each transaction for compliance with semantic integrity rules as much as possible at the global level, and iii) take appropriate actions if violations are detected.

1.0 INTRODUCTION

This report gives elements of a technical specification of an Experimental Semantic Integrity System (XSIS) for a networking environment. It is based on the fundamental concepts of database semantic integrity and on the existing tools and techniques reported by Fong [FONGE 80]. Since this report represents the follow-on work to that cited above on database semantic integrity, basic tutorial material is not repeated here. A glossary is included at the end of this report.

This section summarizes issues of database semantic integrity in a networking environment. Section 2 describes design assumptions and semantic integrity (SI) requirements. Detailed descriptions of selected SI features are provided in Section 3, followed by the overview and design of the software components of XSIS in Section 4. The specification language is then presented in Section 5, followed by concluding remarks in Section 6.

1.1 Semantic Integrity

Data in a database represents an abstraction of a "real world" application. The values stored at any given time represent a particular configuration of that application domain. The problem of SI is ensuring that the data accurately represents the intended meaning of the application at all times. Specifically, SI assures that database interactions are meaningful in terms of correct query formulation and valid update actions.

The extent to which the above assurance can be achieved is fairly limited in today's technology. Current techniques include some simple, adhoc data-type checking, or special purpose audit routines which are usually embedded in application programs. Present capabilities make the design, understanding, analysis, and enforcement of SI difficult. Furthermore, the implementation of any sophisticated SI enforcement is very costly [BADAD 79].

Our approach to enforcing SI within a database management system (DBMS) is to design an SI (sub)system as a self-contained support unit extending the capabilities of a DBMS. (Hereafter, we refer to this subsystem as the semantic integrity system.) As described by Fong [FONGE 80], the issues related to the design of an SI system are:

1. The specification of the SI rules or constraints,

2. The enforcement of those integrity rules, and

3. The actions to be taken when integrity rules are violated.

The specification of the SI rules states explicitly the correctness and consistency of a given database. To enforce such rules, the SI system must prevent transactions from mapping a consistent state to an inconsistent one. The extent and the type of enforcement remain a designer's choice, as does the tradeoff between the desired needs versus implementation and execution costs. Violation actions must be specified when an error condition is detected.

1.2 Semantic Integrity In A Networking Environment

Based upon a literature review [FONGE 8Ø], presently, there is no major work in the area of SI for the networking environment. However, SI is significant in a distributed environment, since local DBMS management is likely to want strong assurances that remote, and therefore, perhaps less knowledgeable users, will not affect database integrity. The major concern is that remote and local updates are semantically correct.

Semantic integrity issues in the networking environment differ in several essential ways from a centralized DBMS. Some of the issues that differ are:

1. Global versus local integrity - In the networking environment there is a two-tier system: global and local levels. Semantic integrity constraints can be applied to the global level in addition to those established by local DBMS management. However, the global constraints must be consistent with those established at each local site.

2. Conflicting constraints - It is possible that constraints appropriate to different local DBMSs may be in conflict. These must be resolved among the local DBMS managements.

3. Remote versus local enforcement - The enforcement of integrity rules are those procedures (computer programs) which insure that the integrity rules specified are not violated. The enforcement program may or may not be executed remotely from the host where the database resides. Deciding where to apply enforcement requires a tradeoff analysis of the amount of the data movement against additional processing by the local host.

4. Partial data view - Some of the local data may not be available to the global user. Thus, it may be impossible to provide global checking which is complete from the local point of view.

2.0 DESIGN ASSUMPTIONS

The XSIS is being designed to operate as part of a distributed environment supporting access to multiple, remote, heterogeneous DBMSs as provided by an Experimental Network Data Management (XNDM) [KIMBS 79]. The assumed XNDM environment and the XSIS design requirements are discussed below.

2.1 The XNDM Environment

The XNDM environment assumes the following.

1. There is a collection of host computers linked together in a communication network.

2. Each or some of the hosts support computation and also support a database for sharing among the network community.

3. The database includes schemas (descriptions) of the database as well as a database management system to maintain and access the local database.

We further assume that each DBMS may have a different data model and may use a different data language. Although each local DBMS and the database are independent and possibly autonomous, these databases are virtually tied together by a higher level schema that enables users of one node to access data that resides in another node without having a detailed knowledge of the specific local schema. The network (global) users view the integrated data as a whole without knowing where data actually reside, whether data are redundantly stored, or without knowing the details of the DBMS's logical data structure.

AD-A102 776 NATIONAL BUREAU OF STANDARDS WASHINGTON D C INST FOR--ETC F/G 9/2
XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER.(U)
JUN 81 S R KIMBLETON, P S WANG, E N FONG F30602-77-C-0066
UNCLASSIFIED RADC-TR-81-123 NL

4 OF 5
AD
A102776

The XNDM has no administrative control over the establishment, organization, or availability of the local databases being shared globally.

2.2 XSIS Requirements

The XSIS checks the SI of all database transactions expressed via the network data language (XNDL). The design of XSIS is guided by following principles.

1. The XSIS assumes that remote users are less knowledgable than local, native mode users. The XSIS also attempts to minimize local system overhead in supporting SI.

2. The XSIS asserts that offloading of SI enforcement from both target systems and their DBMS is desirable. (Local management may be unwilling to incur an extra processing burden to support remote users.)

3. The XSIS enforcement should minimize data movement. Hence, the enforcement procedure should stay, as much as possible, at the processing node rather than at the location of the remote database.

4. The XSIS should be independent of the XNDM and the local DBMS in order to permit a modular design, implementation, operation and modification cycle without impacting local systems or other XNDM components.

The facilities provided by XSIS are the following.

1. Users or network database administrators may state SI rules in a constraint specification language.

2. Specified SI rules are enforced either automatically or under user control.

3. Users are alerted to errors if rules are not satisifed.

3.Ø XSIS FEATURE DESCRIPTION

All of the described SI features reported by Fong [FONGE 8Ø] might be included in XSIS, but in practice some of the integritv checks may be too costly. The XSIS intends to demonstrate the feasibility of certain features in terms of usefulness and operational cost performance. The XSIS SI features are divided into five main categories of which the first three are constraints-related while the last two are system support features:

1. Strong data type constraints,
2. Constrained data dependency assertions,
3. Transition constraints,
4. User-controlled enforcement, and
5. Integrity maintenance facilities.

3.1 Strong Data Type Constraints

Strong data type constraints specify the set of values which instances, variables, and constants of a specified type may assume. They also specified the operations that may be performed on the data type. We have defined a strong data type specification to be composed of four separate constraints: value constraints, extended format constraints, domain compatibility constraints, and legal operator constraints.

Value constraints are those restrictions imposed upon an individual data value such as range (e.g., AGE between 1 to 100), enumerated set (e.g., COLORS are RED, BLUE, and GREEN), uniqueness property (e.g., KEYS must be UNIQUE), and essentiality property (e.g., SOCIAL-SECURITY-NUMBER must be NON-NULL).

Extended format constraints gives in greater granularity, the data type specification based upon the primitive types such as integer, character, or real. For example, the first character of SNO must be the letter "S".

Domain compatibility constraints are violated if two data domains are of different units of measure or are semantically different objects that may not be equated or compared. For example, WEIGHT in pounds cannot be compared with WEIGHT in kilograms.

Legal operator constraints limit the operations which can be performed on a given datum. A set of legal operators may be associated with an item. Attempting to perform any unassociated operation results in an SI violation. For example, SOCIAL-SECURITY-NUMBER must not be algebraically combined with other data.

Among the subcategories of strong data type constraints, we will consider the "uniqueness" property as a column constraint to be described below. The rest of the strong data type constraints are designed to be incorporated within XSIS.

3.2 Constrained Data Dependency Assertions

The constrained data dependency assertions provide database administrators (DBAs) with the ability to express semantic properties of the database, specifically, with the ability to express validity requirements based upon the application environment. These integrity constraints may involve more than an individual datum. The constraint expresses a condition on a subset of the database.

Fong. [FONGE 80] has identified three categories of data dependency: data grouping, aggregate function, and data model constraints. In the context of the XNDM environment, the constrained data dependency assertions are applied to the global schema which is represented in tabular (or relational) form. We can, therefore, consider the data model constraints to be relational. XSIS considers aggregate function constraints to be either as individual data element constraints (if aggregate functions such as average, minimum or maximum values are stored as individual items) or as data grouping constraints (if aggregate functions need to be computed).

As in the case of the XNDM environment, the data grouping constraints can be applied to a tabular representation. The classifications are identified as follows, in increasing order of complexity:

1. Individual data consistency,

2. Row consistency,

3. Column consistency,

4. Intratable consistency, and

5. Intertable consistency.

The individual data consistency constraints are those treated in the strong data type within the value constraints. The row consistency constraints express conditions which involve other values in the same row entry within a given table. Checking requires the retrieval of at least one other data value which occurs on the same row. An example of a row consistency constraint is:

- If DATE-OF-BIRTH and DATE-OF-HIRE are two data items in the same row, the row constraint might be stated that DATE-OF-BIRTH must be less than DATE-OF-HIRE.

The condition for checking column consistency is dependent upon the values in other column entries for a given table. An example of a column consistency constraint might be:

- The SALARY of JOHN should be greater than the average SALARY of all employees.

Although the above example can also be consider as an aggregate function constraint, implementation-wise, this requires retrieving all column values in the salary field in order to compute the average value.

3.3 Transition Constraints

Transition constraints specify conditions that must hold for every update to the local databases. For XSIS, three types of update rules are defined: change rules, insertion rules, and deletion rules. The change rule incorporates the OLD/NEW keywords. Presently, the change rule may modify only one value.

3.4 User-Controlled Enforcement

It is often necessary for the user to control enforcement of specific integrity constraints. This need arises on at least two occasions. Firstly, a single request may not constitute a "complete" transaction. Checking must be deferred until all changes are completed. Secondly, checks might be suppressed for reasons of efficiency in which SI enforcement can be deferred to permit quick responses.

In XSIS, an SI enforcement flag is ON or OFF, which allows a user to declare or suppress enforcement.

3.5 Integrity Maintenance Facilities

The above SI specifications are maintained by XSIS in SI tables which are linked to a network-wide data dictionary via data names. The XSIS provides integrity maintenance commands to the user for:

1. Displaying the integrity specifications,

2. Deleting a specific integrity specification,

3. Defining a new integrity specification, and

4. Changing an existing integrity specification.

The XSIS provides the integrity checks because of the view that SI is a system responsibility. Users are not allowed to code their own integrity checks. The XSIS also maintains timing and frequency statistics for each integrity specification in order to assess cost-performance of enforcement.

4.Ø XSIS SOFTWARE COMPONENTS

This section describes the functional components of XSIS. The XSIS checks remote database operations expressed via XNDM's Experimental Data Language (XNDL). The overview is illustrated by Figure 1. The XSIS is conceived as three separate software components:

1. Specification and maintenance module,

2. Evaluation and enforcement module, and

3. Violation-action processor module.

4.1 Specification and Maintenance Module

The specification and maintenance module permits users to define new integrity rules and to change, delete or display existing integrity rules.

The define function accepts four kinds of constraint specifications: strong data type, data dependency, user-controlled, and transition constraints. (The precise syntax of each will be described later.) These specifications are processed by the specification language processor and generated as entries to the SI tables. The SI tables are linked to the network-wide data dictionary via each data element name.

It is desirable, when defining a new rule to check the collective consistency to all rules. However, the technique for checking consistency of integrity rules is very costly; a practical algorithm does not exist [ZANOG 79].

4.2 Evaluation and Enforcement Module

This module is triggered whenever a database operation is entered via the XNDL. The incoming XNDL (query and update) operation is validated to satisfy specified constraints.

The processing sequence is presented in Figure 2. The XSIS receives a parse-tree representation of XNDL; a detailed description is given by Kimbleton and Wang [KIMBS 80]. Assume the basic structure of the parse-tree token to be given to XSIS as a triplet:

<data name> <operator> <variable>

where <data name> is the name of an element defined in the network-wide data dictionary, <operator> is one of comparison relations, i.e., =, ~=, <, <=, >, >=, and <vairable> is either another data name or a constant.

Many integrity rules can be checked globally and statically without accessing local databases. The goal is to globally check where possible, thus locally checking only those properties that cannot be established statically. The enforcement module, therefore, consists of two phases: a static check of strong data type constraints, and a dynamic test based upon defined assertions.

4.2.1 Strong Data Type Verification - Based upon the data name, the specific strong data type constraint information is located. Then the checking process is performed; it consists of the following four steps.

1. Textual analysis consists of extensive type checking according to the format pattern specification for the data type.

2. Operator analysis consists of checking the given operator which must be among those defined as legal operators for the data name.

3. Domain compatibility analysis consists of checking the given variable to be among the legal domain names for the data name.

4. Value analysis consists of testing for proper ranges of values or legal value among a set of possible values. Notice that the analysis is performed statically with the given triplet. The dynamic checks will be performed later with the assertion verification for individual data item.

If any violation is detected, a condition code is emitted and control is passed to the violation-action processor.

4.2.2 Assertion Verification - After validating the strong data type constraints, the enforcement module enters the assertion evaluation phase. Enforcement is dependent upon the amount of data needed, since XSIS wants to minimize data movement.

Based upon the same data name, the stored assertions are retrieved and analyzed. Inputs to analysis consist of the amount of data needed, the data location(s), and the assertion processing required. When the required data is gathered, the assertion is evaluated. If no violation occurs, control is returned to XSIS which eventually returns to XNDM. If a violation is detected, a condition code is output to the violation-action processor.

4.3 Violation-Action Processor Module

The violation-action processor prints an error message and rejects the XNDL command. (A user-provided error routine could be substituted for this processor.)

5.Ø XSIS SPECIFICATION LANGUAGE

The integrity specification language of XSIS has been designed to facilitate the specification of all four types of integrity features. Emphasis has been placed upon strong data type specification because the specification can be used to accommodate static checking at the global level. Each type supported by the specification language is described below.

5.1 Strong Data Type Specification

The strong data type specification is based upon denotational specification, or in the database context, it is similar to the schema specification [BRODM 78]. The strong data type also supports the notion of abstract data types in the sense that certain manipulation and representation are incorporated in the specification [GUTTJ 77].

The strong data type specification defines a data type by giving common data type properties. The specification consists of five components: (The BNF-like specification appears in Appendix A.)

1. Data name,

2. Format type consisting of either a literal or primitive type followed by the number of occurences of the said type,

3. Legal operators consisting of a set of comparative operators (e.g., = NOT=), and/or a set of arithmetic operators (e.g., + - * /), and/or a set of relational operators (e.g., < > <= >=),

4. Compatible domain list consisting of names of other data elements that are compatible for comparison purposes.

5. Value constraints consisting of a simple Boolean statement, a range statement, an enumerated set, and a non-null indicator.

An error code may be optionally supplied, which will be printed when a violation is detected.

Some examples of strong data type specifications are presented in Appendix B.

5.2 Data Dependency Assertions

As described earlier, we have divided data dependency assertions into individual data consistency, row consistency, column consistency, intratable and intertable consistency. The information needed to check a data dependency assertion is:

1. The constrained data collection, e.g., single, row, column etc., and

2. The assertion stated in predicate form. The specification syntax of the predicate is similar to SEQUEL predicates [CHAMD 76]. The BNF-like specification appears in Appendix C.

5.3 Transition Constraints

The XSIS determines necessary checking when a data value change request is issued. It also checks when a new row is inserted or deleted. For XSIS, three types of transition constraints are defined: change rule, insertion rule and deletion rule.

The syntax of the change rule specification is:

```
OLD <data name> <relation> NEW
```

where <data name> is the name of the data element to be changed, <relation> is the relational operator consisting of =, Not=, <, <=, >, >=. An example of a change transition constraint is:

OLD SALARY < NEW

For insertion and deletion rules, the operation is performed on the entire row. The predicate assertion is associated with the input values for insertion operations expressed in XNDL and, for deletion operations, the predicate assertion is associated with the values to be deleted. The syntax consists of two parts:

1. INSERT or DELETE, and

2. The assertion stated in SEQUEL predicate notation. See Appendix C.

5.4 User-Controlled Enforcement

As described earlier, the user-controlled syntax is simply a flag indicating ON or OFF. The default is ON.

6.Ø CONCLUDING REMARKS

We have investigated the problem of designing a semantic integrity system for database systems in the context of a networking environment. Although the SI techniques used are those that exist for single DBMSs, we have described how those enforcement methods are applicable in the distributed networking environment.

The major purpose of this report has been to provide the elements of a technical specification of XSIS. The principal emphasis has been on the development of a strong data type specification and enforcement facility to permit global checking, thus reducing costs.

This work represents a beginning in developing techniques for providing correct remote interaction with distributed databases. Further research is necessary.

REFERENCES

[BADAD 79] Badal, D. Z., "On Efficient Monitoring of Database Assertion in Distributed Databases," Proceedings of 4th Berkeley Conference on Distributed Data Management and Computer Networks, August 1979, pp. 125-137.

[BRODM 78] Brodie, Michael L., "Specification and Verification of Data Base Semantic Integrity," University of Toronto, Computer System Research Group, Technical Report CSRG-91, April 1978.

[CHAMD 76] Chamberlin, D. D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control, " IBM Journal of Research and Development, Vol. 2Ø, (Nov. 1976) pp. 56Ø-575.

[FONGE 8Ø] Fong, Elizabeth and Stephen R. Kimbleton, "Database Semantic Integrity for a Network Data Manager," Proceeding of National Computer Conference, 198Ø, pp.261-268.

[GUTTJ 77] Guttag, J. V., "Abstract Data Types and the Development of Data Structure," Comm. ACM, 2Ø, 6 (June 1977), pp. 396-4Ø4.

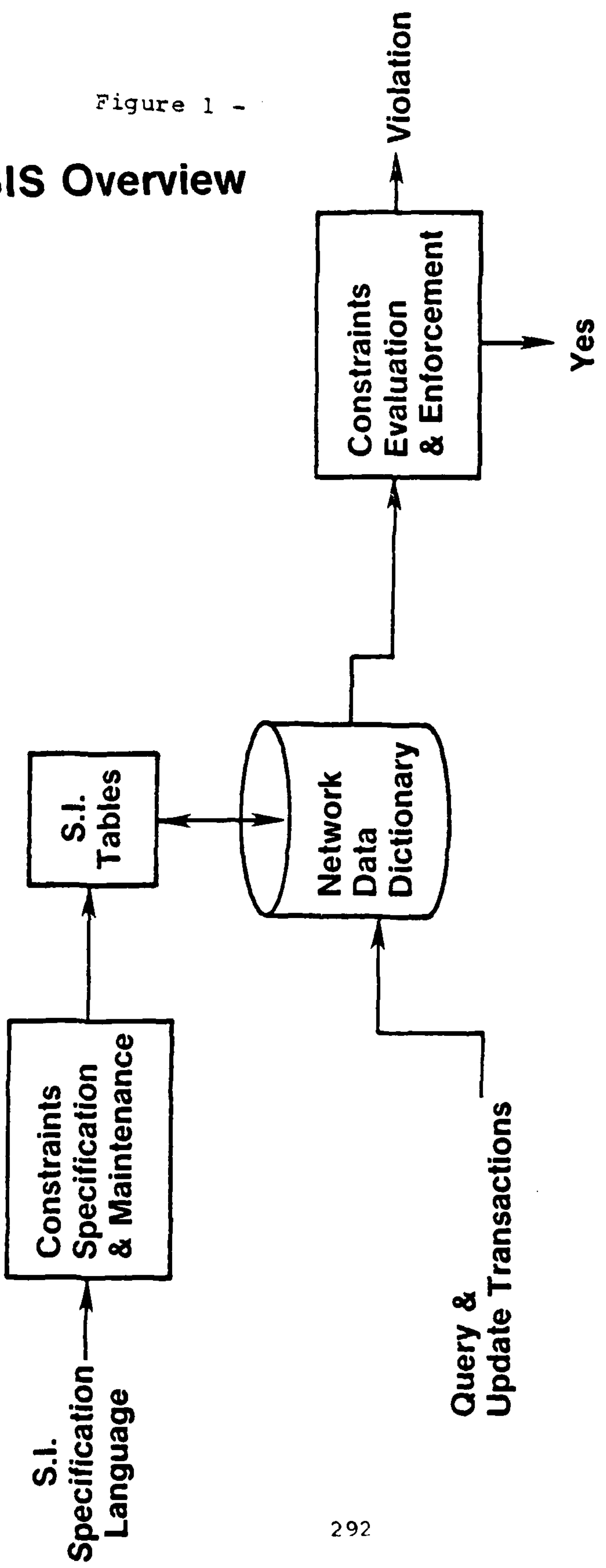
[KIMBS 79] Kimbleton, S. R., Pearl Wang and Elizabeth Fong, "XNDM: An Experimental Network Data Manager," Proceedings of the 4th Berkeley Workshop on Distributed Database Management, August 1979, pp. 3-17.

[KIMBS 8Ø] Kimbleton, S. R. and Pearl Wang, "Applications and Protocols," to appear in An advanced Course on Distributed Systems - Architecture and Implementation," M. Paul and H. Siegert, (Eds.) Springer-Verlag, New York, 198Ø.

[ZANOG 79] Zanon, Guy and Jack Minker, "Consistency and Integrity of Data Bases," TR-723, Computer Science Department, University of Maryland, 1979.

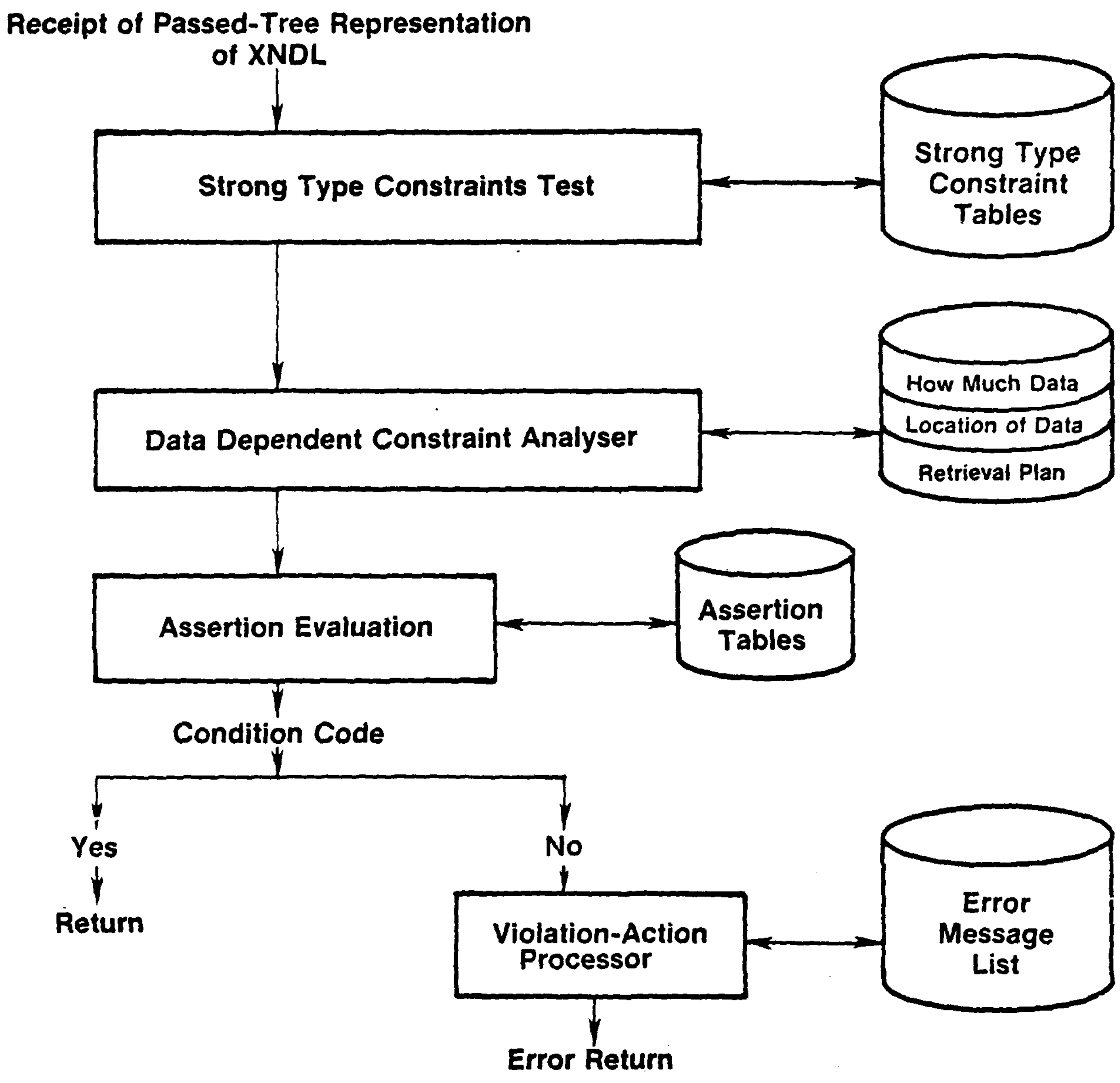
Figure 1 -

XSIS Overview



S.I. Specification Language
Constraints Specification & Maintenance
S.I. Tables
Network Data Dictionary
Query & Update Transactions
Constraints Evaluation & Enforcement
Violation
Yes

Figure 2 - XSIS Processing Sequence



Receipt of Passed-Tree Representation of XNDL
Strong Type Constraints Test
Strong Type Constraint Tables
Data Dependent Constraint Analyser
How Much Data
Location of Data
Retrieval Plan
Assertion Evaluation
Assertion Tables
Condition Code
Yes
Return
No
Violation-Action Processor
Error Message List
Error Return

APPENDIX A - BNF-LIKE SPECIFICATION FOR STRONG DATA TYPE

The BNF-like specification for a strong data type is given below. Square brackets [] indicate optional constructs.

```
     <strong-compound-data> ::= <strong-elementary-data list>

<strong-elementary-data-list>  ::= [<strong-data-type>]
            |<strong-elementary-data-list> , <strong-data-type>

<strong-data-type> ::= <data-name> <format-type> <legal-operator>
                <compatible-domain> <value-constraint> <error>

<data-name> ::= <string>

<format-type> ::= "<literal>"
                 | <type-list>*  END

<type-list> ::= <primtive-type> <no.-of-occurence>

<primitive-type> ::= <single-type>
                    |<variable-length-type>

<no.-of-occurence> ::= <number>

<single-type> ::= <alpha> | <digit>  | <dot>
                 | <dash> | <dollar> | <blank>

<alpha> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
            |a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<digit> ::= Ø|1|2|3|4|5|6|7|8|9

<dot> ::= .

<dash> ::= -

<dollar> ::= $

<blank> ::=

<variable-length-type> ::= <number> | <string>

<number> ::= <digit>*

<string> ::= <alpha>*

<legal-operator> ::= <comp> | <arith> | <relation>
                    | <comp> <arith>  | <comp> <relation>
                   | <arith> <relation> |<comp> <arith> <relation>
```

```
<comp> ::= = | !=

<arith> ::=  + | - | * | /

<relation> ::=  < | > | <= | >=

<compatible-domain> ::= <domain-name-list>

<domain-name-list> ::= <domain-name>
                     | <domain-name-list> , <domain-name>

<domain-name> ::= <string>

<value-constraint> ::= <simple-boolean> | <range>
                     | <enumerated-set> | <non-null>

<simple-boolean> ::= <data-name> <comp-rel> <constant>

<comp-rel> ::= = | != | < | <= | > | >=

<constant> ::= <number> | " <string> "

<range> ::= <data-name> BETWEEN <constant> AND <constant>

<enumerated-set> ::= SET <table-name>

<table-name> ::= <string>

<non-null> ::= NON-NULL

<error> ::= <code> | <error-routine-name>

<code> ::= <number>

<error-routine-name> ::= <string>
```

APPENDIX B - SELECTED EXAMPLE OF STRONG DATA TYPE

A few selected example of strong data type specification is illustrated here in this Appendix. The style of presentation here is for ease of readability.

DATA NAME: moneytype (e.g. $50000)

FORMAT-TYPE:

```
moneytype::= "$" <digit>
```

LEGAL OPERATORS:

```
+, -, *, /, =, NOT=, <, <=, >, >=
```

COMPATIBLE DOMAINS:

none.

VALUE CONSTRAINTS:

```
moneytype BETWEEN Ø AND 999999
```

DATA NAME: nametype (e.g. Elizabeth N. Fong)

FORMAT TYPE:

```
nametype ::= <first> <blank> <middle> <blank> <last>
<first> ::= <alpha>*
<middle> ::= <alpha>* "."
            |<alpha>* "-" <alpha>*
<last>::= <alpha>*
<blank> ::= " "
```

LEGAL OPERATIONS:

```
=, NOT=
```

COMPATIBLE DOMAINS:

namefield

VALUE CONSTRAINTS:

NON-NULL

DATA NAME: ssntype (e.g. 123-45-6789)

FORMAT-TYPE:

```
   ssntype ::= <digit> 3 "-" <digit> 2 "-" <digit> 4
   <digit> ::= 0|1|2|3|4|5|6|7|8|9
```

LEGAL OPERATORS:

=, NOT =

COMPATIBLE DOMAINS:

none

VALUE CONSTRAINTS:

ssntype NOT= 000-00-0000

DATA NAME: addresstype (e.g. 123 Long Road, Gaithersburg, MD, 20760)

FORMAT-TYPE:

```
  addresstype ::= <no> " " <road> "," <city> "," <state> "," <zip>
  <no> ::= <digit>*
  <road> ::= <alpha>*
  <city> ::= <alpha>*
  <state> ::= <alpha> 2
  <zip> ::= <digit> 5
```

LEGAL OPERATORS:

none

COMPATIBLE DOMAINS:

none

VALUE CONSTRAINTS:

<state> IN SET statecode.

APPENDIX C - SUBSET OF SEQUAL PREDICATE SYNTAX

```
SEQUAL predicate ::= <simple-boolean>
                   | IF <pred> THEN <pred>
                   | <expr> BETWEEN <expr> AND <expr>
                   | <expr> <comparison> <table-spec>

<simple-boolean> ::= TRUE |FALSE
                   | <pred>

<pred> ::= <data-name> <comparison> <constant>

<data-name> ::= <string>

<comparison> ::= = | NOT= |< | <= | > | >=

<expr> ::= <data-name>
         | <constant>

<constant> ::= " <string> "
             | <number>

<table-spec> ::= <string>

<string> ::= <alpha>*

<alpha> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
           |a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<number> ::= <digit>*

<digit> ::= Ø|1|2|3|4|5|6|7|8|9
```

GLOSSARY

ASSERTIONS - Constraints that are applied to actual values of the database.

ATTRIBUTE - A category of identifier or label applied to an entity. For example, the entity CAR has the attribute COLOR which may have the value "BLUE".

CODASYL MODEL - A type of network model defined and maintained by the Data Base Task Group (DBTG) CODASYL committee.

CONCURRENT UPDATE - Multiple users updating a shared database.

CONSTRAINT - The validity criteria of the properties of database that are expressed explicitly using data definition and manipulation languages.

DATABASE - A collection of data values that are accessible through computer programs.

DATA TYPE SPECIFICATION - A prescription of both the logical and physical form of each data value of the data type.

DATABASE MANAGEMENT SYSTEM - An integrated set of computer programs that collectively provide all the capabilities required for centralized management and control of access to a database that is shared by many users.

DATA ELEMENT - The smallest named logical unit of data.

ENTITY - A primitive term used to identify a person, object, idea, event, or thing that is the subject of data collection.

INTEGRITY - The condition of a database in which the data is accurate and consistent.

NETWORK MODEL - A data model whose pattern of organization is based on superior and inferior relationships of data, where an inferior record may have more than one immediate superior.

RECORD - A collection of logically related data elements representing a composite unit of information within a file.

RELATIONAL MODEL - A data model whose pattern of organization is based on a collection of tables of data elements; each table represents all occurrences of a relationship.

RELATIONSHIP - A correspondence or association between or among sets of entities.

RELIABILITY - The measure of hardware or software functioning correctly.

RETRIEVAL - The process of obtaining stored data from a database. The process includes the operations of identifying, locating, and transfering the data.

SCHEMA - A complete description of a database, written in a data definition language, giving the characteristics of the data and the relationships between the units of data.

SEMANTIC INTEGRITY - The extent to which a database accurately represents particular properties of the intended application. Operationally, semantic integrity is to ensure that all inputs and updates are valid against constraints.

STRONG DATA TYPE - An extended data type specification to include more logical properties of data. The properties may be in syntactic specification and the abstract meaning of the operations associated with a data entity.

UPDATE - The process of changing, adding, or deleting one or more data values stored in a database.

VALUE - A text string, numerical quantity, Boolean value, etc. that can be assigned to an attribute.

VERIFICATION - The enforcement or check of the satisfiability of the integrity constraint against a data request.

APPENDIX A

REMOTE RECORD ACCESS: REQUIREMENTS, IMPLEMENTATION, AND ANALYSIS

Helen M. Wood

Stephen R. Kimbleton

National Bureau of Standards

August 1979

ABSTRACT

A key support component for network-wide data sharing is the ability of a process to access remotely stored data at runtime. In order for the accessed data to be useful, a means of overcoming differences in data representation and format is necessary. Such a capability is termed remote record access. This paper identifies some of the problems inherent in the sharing of data among dissimilar computer and data systems. Implementation issues and alternatives are presented, followed by a description of XRRA, the Experimental Remote Record Access component which has been implemented as part of the Experimental Network Operating System (XNOS) at the National Bureau of Standards.

1. INTRODUCTION

The emergence of computer networks from the research stage to the production environment has been accompanied by a growing need to buffer the network user from the components of the network. Such a buffer would mask the differences between computer systems (hosts) on a network, thus allowing network users to spend less time learning the idiosyncracies of each system and more time utilizing network services. Network Operating Systems (NOSs) [KIMBS 76, 78], [FORSH 78] are intended to provide this type of buffer by supporting and simplifying access to existing services. In addition, NOSs could expedite the construction and subsequent accessing of new services by simplifying interaction among systems and between systems and users.

Crucial to the realization of NOS objectives are the abilities to (1) exchange data between cooperating (but not necessarily colocated) processes, and (2) preserve the meaning, and hence the usefulness, of that data. Traditionally, when it was known that a program on one computer system would require data from another, a decision was made to colocate the program and data on whichever system would require the least effort and expense. Although for certain high bandwidth applications, colocation may still be preferrable, the increasing size and complexity of programs, files, and data bases, coupled with the often rapid response time requirements for information, make such an approach insufficient. The ability for a process on one machine to access and make use of data on another at run-time thus has become a prerequisite for realizing the full potential of computer networking. Such a capability is termed Remote Record Access (RRA).

This paper discusses the issues and alternatives related to the implementation of a remote record access capability. The remainder of Section 1 identifies goals of and solution requirements for a remote record access facility. Section 2 considers the data conversion problem in depth, and includes descriptions of related efforts. Section 3 identifies various structural considerations including the functional and information requirements and architectural alternatives involved in implementing an RRA capability. The NBS implementation of the Experimental Remote Record Access component (XRRA) is then presented, followed by a discussion of RRA in the context of higher-level, communications protocols.

1.1 RRA Objectives

A basic design objective for a RRA service is providing process independence from data location and originating format. It is envisioned that a RRA capability would be of most use in support of network access to Data Base Management Systems (DBMSs) and exception reporting systems (i.e., low bandwidth applications, as previously mentioned).

Location transparency seems a fairly straightforward, bounded problem primarily requiring a source of knowledge about network-wide resources (e.g., a network resource directory). Data format independence, however, may not be nearly so feasible if the range of support is not carefully specified.

When discussing protocols for data sharing, Kimbleton [KIMBS 78] noted that data transfer protocols can be distinguished by three levels of difficulty, depending on whether the block of data is generated by: i) a given data element type (e.g., characters), ii) a pointer free structure (e.g., a COBOL record), or iii) a structure containing pointers.

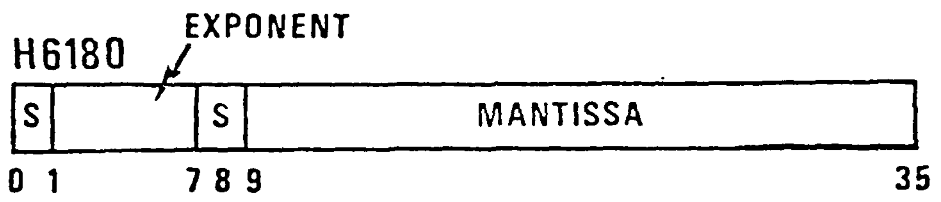
Case (i) is clearly feasible, as this is the case supported by the ARPANET File Transfer Protocol (FTP). Case (ii), however, is significantly more difficult. A description of the structure's graph is required, along with an identification of the structure's data elements, the mapping between structures, and complex programs to manipulate this information. The examples of real and character data representations, shown in Figures 1-1 and 1-2, are indicative of the complexity of the problem at just the data-type level.

Supporting data independence for structures containing pointers (case iii) is likely to prove extremely difficult. This is primarily because of the architectural dependence which can exist between the interpretation of the pointer and its representation. It should be noted during this discussion, that if host access methods are used to retrieve data, then any physical incompatibilities due to secondary storage formats (e.g., blocking factor) need not be considered.

This approach is therefore compatible with the concept of "protocol" as set forth by Crocker [CROCS 72]:

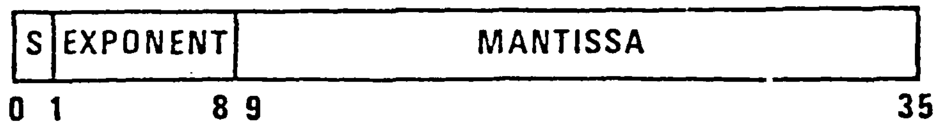
> When we have two processes facing each other across some communication link, the protocol is the set of their agreements on the format and relative timing of messages to be exchanged. When we speak of a protocol, there is usually an important goal to be fulfilled. Although any set of agreements between cooperating (i.e., communicating) processes is a protocol, the protocols of interest are those which are constructed for general application by a large population of processes in solving a large class of problems.

H6180



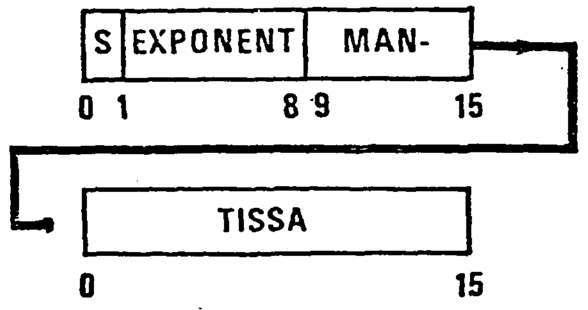
- EXPONENT 2'S COMPLEMENT
- MANTISSA 2'S COMPLEMENT

DECSYSTEM-10



- EXPONENT IS 1'S COMPLEMENT, EXCESS 200_8 CODE
- MANTISSA 2'S COMPLEMENT
- NEGATIVE MANTISSA FORCES NEGATIVE EXPONENT
- MANTISSA SIGN IS BIT 0.

PDP 11/45



- EXPONENT IN EXCESS 200_8 NOTATION
- MANTISSA IN SIGN-MAGNITUTE NOTATION
- MOST SIGNIFICANT BIT OF MANTISSA NOT STORED.

FIGURE 1-1:
FLOATING POINT REPRESENTATIONS OF REAL DATA

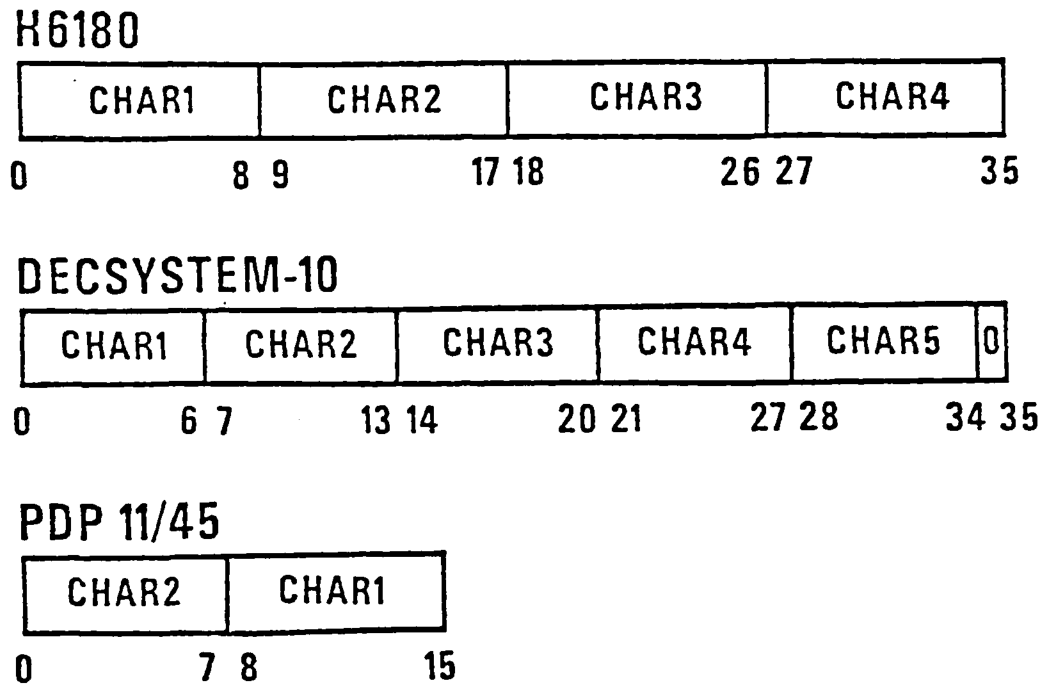


FIGURE 1-2:
CHARACTER REPRESENTATIONS

The important point here, besides the generally acceptable definition of protocol, is that such tools are for "general application" by a "large population of processes" which are used in solving a "large class" of problems. Since RRA has many of the characteristics of a protocol (cf. Section 6), an approach to RRA which emphasizes breadth rather than depth (in the data conversion area) seems to be the proper alternative. Therefore, based on the above considerations, it seems desirable to confine the scope of an RRA capability to cases (i) and (ii) in the context illustrated in Figure 1-3.

Among other desirable characteristics of a RRA facility are flexibility, expandability, minimal host overhead, minimal transmission overhead, and reliability. Clearly, all of these cannot be achieved in an absolute sense in any one implementation. The development of a RRA prototype can, however, provide a wealth of substantive information that can assist in evaluating the cost benefits of supporting such capabilities in a specific applications environment. Furthermore, such an effort can assist in the identification and development of appropriate standards for the exchange of structured data in distributed systems. For these reasons, the Experimental Network Operating System (XNOS), developed at NBS, is being utilized in exploring the basic issues in promoting more effective sharing of network accessible resources [KIMBS 78].

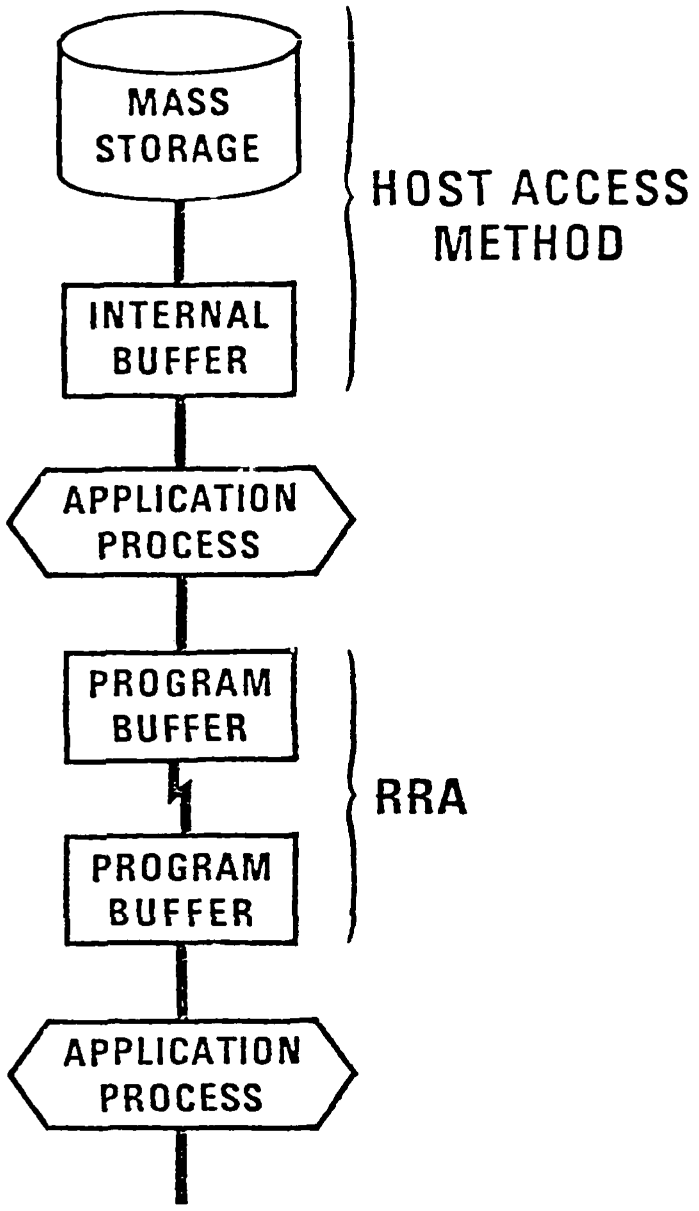


FIGURE 1-3:
RRA SCOPE

While the remainder of this paper will discuss RRA within the context of the NBS XNOS implementation, it should be noted that the functionality of the solution approach applies to the general class of NOSs represented by the NBS system.

1.2 Solution Requirements

In order to provide a remote record access capability, the desired data must be (i) located, (ii) accessed and (iii) any data representation incompatibilities must be resolved. The first requirement involves the specification of the host, user account (e.g., directory), file, and specific record (e.g., via access key) desired. To satisfy the second need a selection process must be available to service the request (e.g., a user program or DBMS). The support mechanisms needed to intercept a program's request for data, activate a process on the host maintaining the data to retrieve the desired record, and return the translated/transformed record to the requesting process must be built upon a protocol which supports network interprocess communication (IPC). Meeting the last specification requires sufficient information to describe the data formats, representations, and the mapping between formats, plus a transformation process to effect the data mapping.

Network Operating Systems provide a useful collection of many of the mechanisms needed to implement a RRA mechanism. Initially we assume a NOS environment as described by Kimbleton [KIMBS 77,78]. NOSs are commonly viewed as the means for masking system differences from users. The functional objective of a NOS is to support and simplify access to existing services and to expedite the construction and subsequent accessing of new services by simplifying interaction among systems and between systems and users.

A major design goal for implementing a NOS on an existing computer network is that the NOS is transparent to the participating host systems. This goal is achievable through a consolidation of NOS support functions into a Network Interface Machine (NIM), as suggested by Kimbleton [KIMBS 76]. The NIM is, in fact, a focal point for user-system and system-system interactions. It serves, among other things, as a translator for commands (e.g., MOVE <file>, DELETE <file>), a transformer for data flowing between network processes, and a source of knowledge of network resources (e.g., maintains a network-wide file directory). The first role provides the NOS user with a standardized view of network resources by supporting a comon command language for all participating hosts [FITZM 78]. The second role is actually that of the RRA component.

NBS developed XNOS to demonstrate the feasibility of such general purpose NOSs and to facilitate the investigation of the capabilities and limitations inherent in such systems. Figure 1-4 illustrates the user view of the network, while Figure 1-5 identifies the current XNOS configuration. Section 2 presents an in depth look at the problem of resolving data incompatibilities. Sections 3 and 4 discuss RRA in a NOS environment in some detail.

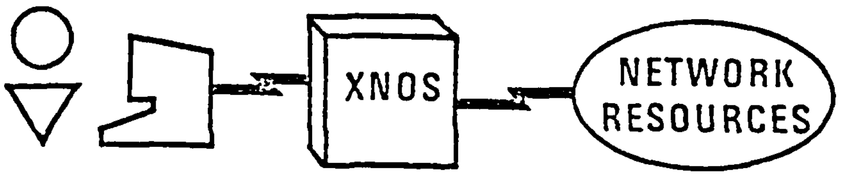


FIGURE 1-4:
USER VIEW OF NETWORK

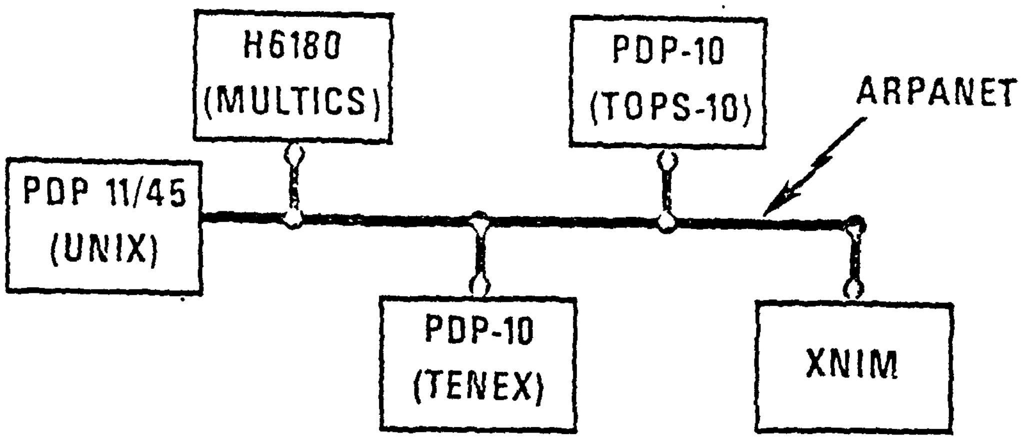


FIGURE 1-5:
XNOS INITIAL CONFIGURATION

2. DATA CONVERSION

Incompatibilities of dàta representation and format are problems that pre-existed computer networking. This is attributable not only to differences in data record format, but to the total lack of industry standards for the internal representation of information in computers.

The continuing need and desire to exchange computer-readable information has given rise to numerous data representation standards including for example the American Standard Code for Information Interchange (ASCII) [ANSI 1,2], the Standard for Bibliographic Information Interchange on Magnetic Tape [ANSI 4], and the Standard Representation of Numeric Values in Character Strings for Information Interchange [ANSI 3].

In recent years, numerous efforts have been made to automate the process of transforming data. We shall now briefly describe several approaches to solving the translation/transformation problem implicit when data is shared among dissimilar hosts. It should be noted that, as presently configured, none of these systems supports run-time record translation/transformation. That is, the required support mechanisms do not currently exist to facilitate the execution-time binding of host/data names in response to a request by a program for remotely stored data. Instead, these approaches are intended to be invoked by the user directly, rather than by a process acting on the user's behalf, with the source and target data files/bases prespecified. Nonetheless, a consideration of these approaches serves to "set the stage" for identifying the issues of and requirements for the data conversion component of remote record access. (Several of these approaches are compared and contrasted in a recent internal NBS report by Fry [FRYJ 78].)

After discussing these approaches to the data conversion problem, major features of the data conversion portion of the XRRA utility are described.

2.1 Brute-Force Approach

In the past, "brute-force" or "manual file conversion" has been the method used most often to attack the data translation problem. Thus when data in format A needed to be transformed into format B, a special purpose program was written to perform that specific transformation. Although this approach might seem acceptable for sharing data between two systems, when the number of systems increases the problem soon gets out of hand. For example, if one wished to share data between N systems, each requiring a different data format, then (N-1) translators would be needed at every host involved. Alternatively, a centralized data conversion service would have to maintain N(N-1) translators. The need for more general-purpose translation/transformation routines is obvious.

2.2 Generalized Approach

Within the past few years, several methods for attacking the data translation/transformation problem in a more general fashion have been suggested. Common to all of these efforts is a degree of generality and a "descriptive approach" which utilizes descriptions of the source and target data formats and a definition of the mapping to take place [BIRSE 76]. Among other factors, these generalized translation techniques can be categorized by the implementation approach adopted. For the interpretive approach a generalized processing program is developed; while a specific translation program is created for each conversion in the generative approach. Of course, some systems may involve a combination of the two.

2.2.1 DRS. One such conversion system, the Data Reconfiguration Service (DRS), was implemented on the ARPANET [HARSE 71,72], [ANDER 71], [CERFV 72] The DRS allowed the user to specify the transformations to take place on data records (even to the bit level) through the use of a fairly complex, low-level syntax. The resulting module or "form" is essentially a "black-box" that is interjected into the communications path between user and server processes. As described in [ANDER 71]:

> The DRS attempts to provide a notation for form definition tailored to some specifically needed instances of data reformatting. At the same time, the DRS keeps the notation and its underlying implementation within some utility range that is bounded on the lower end by a notation expressive enough to make the experimental service useful, and bounded on the upper end by a notation short of a general-purpose programming language.

The following sequence of DRS statements illustrates a form which could be used to delete 8 bits preceeding a character string [ANDER 71]:

```
(B,,8),            /*isolate 8 bits to ignore*/

SAVE(,A,,10)       /*extract 10 ASCII characters from input stream*/

:(,E,SAVE,);       /*emit the characters in SAVE as EBCDIC characters whose
                   length defaults to the length of SAVE (i.e., 10), and advance to
                   the next rule*/
```

Such forms are used to drive a software module, called the Form Machine, which performs the specified transformation on the data stream. As shown in Figure 2-1, the DRS provides centralized transformation support.

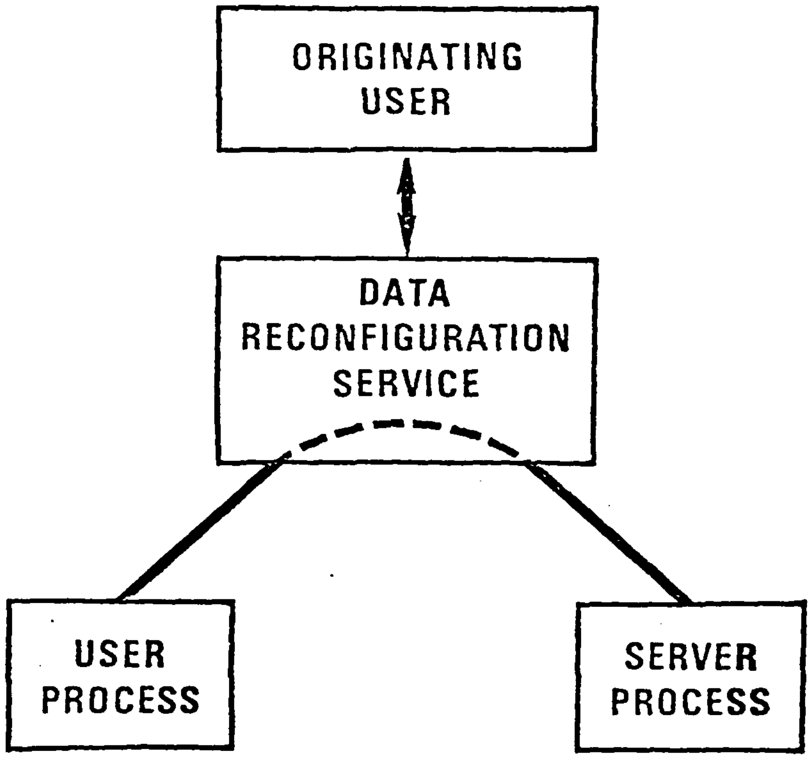


FIGURE 2-1:
DATA RECONFIGURATION SERVICE

One obvious advantage of this approach is the low data transmission overhead incurred vs. that result when a standard, perhaps character-based, format is used to communicate with the data convertor [see Section 3.1.1]. On the other hand, a clear disadvantage is the need to anticipate all needed transformations from M source formats to N target formats and provide the resulting (M x N) transformers to the DRS.

2.2.2 DSCL. The Data Specification and Conversion Language (DSCL), formerly entitled the File Translation Language (FTL), originated as an attempt to solve the same problem areas as DRS, but through use of a higher-level, special-purpose programming language which operates on data viewed as strings of bits. DSCL programs include a DECLARATION SECTION, in which input and output formats and representations are specified, and a

PROGRAM SECTION containing the executable statements. The flexibility provided by this higher-level language approach is evident from the example input/output declaration statements shown in Figure 2-2 [SCHNG 75A]. Here global primitives are used to define concepts such as ASCII, WORD SIZE, and CHARACTER. In addition, automatic services are provided. For example, code conversion is performed automatically whenever the declared input and output code sets of character data items taken from the input source and directed to the output set differ. Thus, in this example, the input data stream would be converted from ASCII to FIELDATA encoding.

```
INPUT

     CODE SET IS ASCII
     WORD SIZE IS 16
     DEFAULT MAPPING IS BEGIN '['=>'(';']'=>')';ALL=>'?'END
     RECORD SIZE IS VARIABLE
     EOR CHARACTER IS CR
     INTEGER REPRESENTATION IS (16,2)

OUTPUT

     CODE SET IS FIELDATA
     WORD SIZE IS 36
     RECORD SIZE IS 112 WORDS
     EOF CHARACTER IS '@'
     COMPRESSION-FLAG IS '1'B
     COMPRESSION-COUNT IS NEXT - TAB
     INTEGER REPRESENTATION IS (36,1)
```

FIGURE 2-2:
DSCL DECLARATION SECTION

It is envisioned that DSCL-like translation services could be centralized, as is the case for the DRS. Thus one machine could in effect become a network translator with DSCL used for all communications. The central machine would maintain the (M x N) translation programs required to support transformations between M source and N target data formats [SCHNG 75A,B].

As with DRS, the DSCL approach has the advantage of a low transmission overhead requirement, plus the provision of a higher-level language. However, (M x N) conversion programs are still required.

2.2.3 SDDL. A major area of data base research is concerned with the problem of data base transformation [FRYJ 72A, 72B, 74], [MERTA 74], [SMITD 72], [BACHM 79], and [SHUN 77]. The University of Michigan has developed an interpretive translation technique which utilizes a stored-data definition language (SDDL) to describe the source and target data bases and a translation definition language (TDL) to define restructuring transformations [FRYJ 72A, 72B, 74]. Compilers for these languages are used to produce tables of parameters which are input to a generalized translation algorithm. Although not specifically designed to support network sharing of data, if the SDDL approach were applied in a networking environment, as illustrated in Figure 2-3, it has been suggested that a single translation program could be required at each of the K hosts supporting shared data on the network. In addition, each host would also need to maintain an SDDL table describing its data and (K-1) TDL tables [BIRSE 76]. (Of course, a centralized approach could also be adopted.) Clearly, as this system was developed to solve the very difficult problem of data base conversion, it would impose a very high overhead burden when only simple data transformations are required.

2.2.4 EXPRESS. Two high level languages have been developed to support data translation [SHUN 76]. DEFINE is a non-procedural data descriptive language and CONVERT is a very high level, non-procedural language designed to operate on hierarchical data. In order to use these languages, input data must first be available in a normalized form. A prototype data translation system, driven by these two languages, has been developed [HOUSB 77]. Possible applications of the EXPRESS system include data base conversion and use with a centralized data base system. While able to handle highly complex transformations, just as for SDDL, this approach would impose a heavy overhead on transactions involving only simple data transfer.

2.3 NBS Record Translator/Transformer

The data conversion component of the NBS XRRA implementation is the Record Translator/Transformer (RTT). RTT is a generalized, non-procedural, table-driven system consisting of two modules. The first is a Record Data Translator (RDT) which performs translations between host native formats and a character-based, intermediate data format, termed Network Normal Form (NNF). The second module is the Record Transformation Routine (RTR) which performs operations on data fields within the record in order to map the incoming data to the format required by the requesting process. Tables are used to supply RTT with descriptions of the input and output data record formats, the native formats of supported network hosts (e.g., bit configuration for INTEGER data), and the mapping required to transform input records into acceptable output record formats. A more detailed description of RTT is contained in Section 4.2.

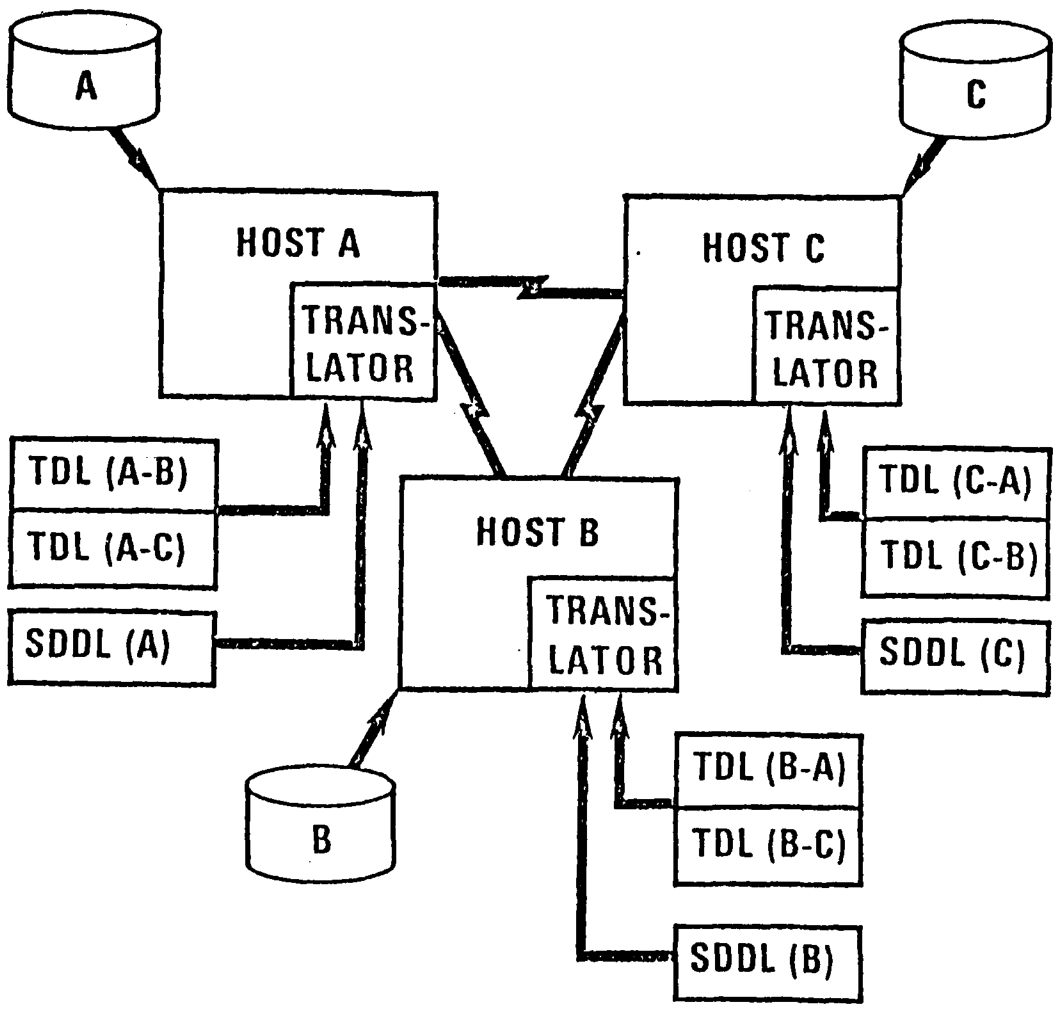


FIGURE 2-3:
SDDL APPROACH IN NETWORKING ENVIRONMENT

2.4 Problems of Data Transformation

Isomorphism is a natural objective in any data translation effort. If A is a record on host A which is translated into a host B representation, then we say that the translations from host A format to host B format and back again define an isomorphism provided that the original record and the record resulting from the two translations applied in succession are identical. Unfortunately, when data is exchanged by systems that support different data formats and representations, data translation problems occur which prohibit isomorphism. A common problem is loss of precision due to varying host word sizes. Other problems include format incompatibility and data type incompatibility. In the remainder of this section we shall briefly describe each of these problems. Levine examines these data translation problems in more detail.[LEVIP 77].

2.4.1 Loss of Precision. Precision is defined as a measure of the degree of exactness with which a quantity is stated [SIPPC 72]. It is a relative term in that it is concerned with the range of values that can be represented rather than with absence of error (i.e., accuracy). Loss of precision occurs, for example, when a data item is moved from a high precision format to a low precision format. Thus, attempting to represent a 32 bit integer in a host which only has 16-bit integer formats results in a loss of precision.

Precision problems cannot be avoided in a heterogeneous environment. Different word sizes and field sizes (e.g., mantissa and characteristic for real data) are the rule rather than the exception. One system may allow only single precision floating point (real) data. Another may not support floating point at all. This is not to imply that data cannot still be usefully exchanged among such systems. However, conventions must be adopted for recognizing such problems and notifying the user/server processes, as appropriate.

2.4.2 Format Incompatibilities. When describing problems with data translation, Levine notes that "..format incompatibility problems occur when data items of a particular type and in a particular format must be translated into a different format for the same type." Unlike the case of precision problems. however, format incompatibilities are strictly a function of the formatting scheme. They do not derive from the range of values (e.g., number of bits) allowed for an item's representation [LEVIP 77]. This problem is best illustrated by noting that the decimal fixed point number 0.2 cannot be exactly represented in binary. Here the transformation from decimal to binary has resulted in a change of value.

As with precision problems, format incompatibilities are unavoidable in a heterogeneous environment. Translators cannot help but introduce errors due to rounding and truncation of numeric data. Ideally, however, users will be informed of the translator's "policy" in dealing with such situations.

2.4.3 Data Type Incompatibilities. Data type incompatibility results when an output format does not exist to receive a given data type. One example of such a situation occurs when a process attempts to output floating point information to a terminal device (i.e., no floating point-to-character transformation has taken place). While there might be a requirement for the provision of some type of terminal handling intelligence to interface a "dumb" terminal to a "smart" network, it is still entirely possible that data type incompatibilities will occur even between other "smart" systems on a network. Therefore, some sort of error detection and recovery mechanisms must be provided to handle such cases.

3. STRUCTURAL CONSIDERATIONS

Although it is the consensus that, especially in a computer networking environment, the generalized approach to data conversion is preferrable to the alternative (i.e., brute-force), there is little agreement on a "standard" method for implementing such systems. Since the organization of a remote record access system has direct impact on the set of support requirements, this section will highlight some of the organizational alternatives and their related considerations/implications, after first identifying the support requirements common to all approaches.

3.1 Support Requirements

Based upon careful consideration of the problem, along with existing solution approaches to the several problem components, it is apparent that solving the remote record access problem requires certain easily identifiable functional and informational capabilities. Providing these capabilities, in turn, gives rise to additional needs (e.g., interprocess communication, arbitrary precision arithmetic capability, specification of a standard data format). These and other support requirements are now discussed.

3.1.1 Functional. The provision of a remote record access capability requires (1) a mechanism for selecting a record from the file/data base containing it, (2) a record translator to preserve meaning in transmitting the record between dissimilar hosts and (3) a record transformer to permit the alteration of record structures.

The precise mechanism which supports record selection is dependent upon capabilities existing at the host computer, including those provided by a data base management system, if any. It is assumed, from the perspective of specifying a RRA capability, that the selector process exists and is capable of retrieving a record based on utilization of a unique key, if random access techniques are employed. The keyword NEXT must be used if sequential access is being supported.

Record translation preserves the logical record structure and data element type (e.g., real, binary, logical, integer, character) and, for arithmetic data elements, precision. Clearly a record translator must know the exact format of the record to be translated, down to the data item, along with the internal format of all data types for each and every system supported.

Record transformation supports modification of both the logical structure of the record and individual data elements. Such transformations are useful in matching the information transmitted to the needs of the receiver (e.g., field reordering). They may also be utilized in controlling access to sensitive information (e.g., by omitting sensitive information from the record before transmitting it on to the requesting process). Such transformation affects the logical structure of the record through one of three basic transformation types: logical, arithmetic, or string. Among the additional transformations that may be needed are algorithms for the compression and/or decompression of textual information, as well as for field or record level encryption/decryption.

The operations currently implemented in XRRA are shown in Table 3-1. The logical transformations AND and OR generate Boolean binary strings resulting from the bit-by-bit ANDing and ORing of two successive strings. The basic arithmetic transformations +,-,/,* act as would be expected. String transformations can be quite complex as evidenced by the capabilities of string manipulation languages. Initially, a concatenation capability is supported.

		DATA TYPE				
OPERATION	SYMBOL	INTEGER	REAL	CHAR	BINARY	BOOLEAN
ADD	+	X	X			
SUBTRACT	−	X	X			
MULTIPLY	×	X	X			
DIVIDE	/	X	X			
AND	&				X	X
OR	|				X	X
CONCATENATE	#			X		

TABLE 3-1:
XRRA TRANSFORMATION OPERATIONS

3.1.2 Information. As shown above, mechanisms for solving the data conversion problem require information about the physical and logical characteristics of the data. Such information must be provided explicitly since, generally speaking, strings of bits do not carry with them any indication of the data type(s) they are representing. Levine [LEVIP 77] notes that this is because "...the overwhelming majority of currently available computer systems are based on the Von Neumann philosophy for storing digital information." Consequently, the semantic meaning of bit strings is derived from the context in which they are used.

Physical characteristics are the actual bit configurations of each type of data maintained on the system. For example, floating point words on the DECSYSTEM-10 are 36 bits in length, have a sign-bit located in bit position 0, followed by an 8-bit exponent in one's-complement, excess 200 (octal) notation, which in turn is followed by a 27-bit normalized mantissa in two's-complement representation. Similar information is also required to fully describe the DECSYSTEM-10's internal representation of integer, character, logical, and Boolean data types. In XRRA, this information is maintained in the Host Representation Table (HRT). Table 3-2 illustrates HRT entries describing the format of real data for three computer systems. However, these descriptions would not be complete for all systems. For example, in the Burroughs B5500, B5600, and CDC 6000 Series computers, the radix point is at the right of the mantissa, rather than the left as for the systems in this table. Also, the IBM 360-370 Series represents the exponent in base 16, rather than base 2. (A good discussion of the plethora of data type representations can be found in [TREMJ 76].)

	DECSYSTEM-10	HONEYWELL 6180	PDP-11/45
WORD SIZE	36	36	16
MANTISSA SIGN LOCATION	0	8	0
MANTISSA BIT POSITIONS	9-35	9-35	9-15, 0-15
MANTISSA MOST SIGNIFICANT BIT STORED	YES	YES	NO
MANTISSA NORMALIZED	YES	YES	YES
MANTISSA REPRESENTION	2'S COMP	2'S COMP	SIGN-MAGNITUDE
EXPONENT SIGN LOCATION	N/A	0	N/A
EXPONENT BIT POSITIONS	1-8	1-7	1-8
EXPONENT REPRESENTATION	1'S COMP	2'S COMP	1'S COMP
EXPONENT EXCESS CODE	128	0	128

TABLE 3-2:
HOST REPRESENTATION TABLE FOR REAL DATA TYPE

A complete description of the organization of the data to be transferred would, on the otherhand, include such information as size of record, names of data elements (also called fields or items), and the type and size of each item. Thus, a description of a data record might look somewhat like a conventional FORTRAN format statement (e.g., 3A5,2X,5I,2X,F7.2), with names associated with each field or, more likely, resemble a COBOL data description. Whatever the form, a need exists for a language to fully describe the data - a Data Description Language (DDL).

In XRRA, a Logical Record Description (LRD) contains such information as the record length and a set of Data Element Descriptions (DEDs) which, for each data element, specify the element level (node), name, and attributes (data type and size).

3.1.3 Standard Data Forms. Although not a prerequisite for data translation/transformation, as a practical matter the use of an intermediate "standard" or "normal" notation to represent the data is desirable. Not only would such a notation reduce the complexity of a general translation algorithm (e.g., Michigan's SDDL approach [FRYJ 74]), but for systems like DRS [ANDER 71] or DSCL [SCHNG 75A,B] where the network translation support is centralized, the number of translation routines would be reduced from N(N-1), for N computer/data systems, to 2N (with one algorithm defining the transformation to "normal" form, and one defining the inverse).

The use of a standard intermediate form for representing data exchanged between potentially different systems is not new. The ARPA protocols TELNET and FTP both support such a convention. The TELNET protocol provides terminal users the means of accessing remote systems as if the user were a local user of that system. Implementation of the TELNET protocol is based on a Virtual Terminal Protocol defining a network-wide set of terminal functions and character encodings. The source computer (system to which the user is logged in) maps the functions and character encodings which it uses into the corresponding VTP functions and encodings. The destination computer (remote system being accessed by the user) maps from these VTP functions and encodings into those which it supports. The ARPA File Transfer Protocol (FTP) also follows this general approach of mapping from a local host representation to a common network representation and back to a local host form. At present FTP only supports transmission of character or binary (i.e., unmapped) files.

Levine [LEVIP 77] examines the use of a standard, character-based format, i.e., characters used to represent all data types, vs. a data format consisting of a standard character set for character data and a set of more data compatible formats for other data (e.g., a non-character based format for exchanging integer data and another for real data). It is apparent that any one approach would not be optimal for all applications. Transmission and processing overhead are certainly among the major factors to be considered when choosing a standard format. For example, if large amounts of non-character data are to be transmitted in a character-based normal form, then there may be both communications bandwith and processing overhead concerns. On the other hand, many processors currently support internal translation to ASCII (and the reverse) in order to communicate with their various terminal and other peripheral devices. Thus, looking ahead to the day when heterogeneous systems exchange structured data in a standard format, a character-based canonical form is likely to be an acceptable compromise and may even be the best general purpose alternative.

The RTT component of XRRA utilizes an ASCII-based intermediate, Network Normal Form (NNF). In this format, all data (even numeric) is represented in a character form. For example, a data field containing data of type REAL would be expressed as

$$\{+,-\}\{d_1,d_2,d_3,...\}.\{d_1,d_2,...\}$$

where each element "dn" is a decimal digit. Binary data, on the otherhand, would be expressed as strings of the ASCII characters "0" and "1". The logical data types TRUE and FALSE appear as "•T•" and "•F•", respectively. Field delimiters may be any character that does not appear within the data fields (e.g., '!'). The following is an example XRRA record in NNF:

!WIDGITS!-03.5686!+32.456!-15!0111001111111001!•T•!

The exchange of self-describing data, in which canonical data descriptive tags accompany the data in its travels (i.e., self-describing records), has also been suggested. A standard format for the exchange of structured data, which employs a data element tag based, data description format is now being proposed by the American National Standards Institute (ANSI) [ANSI 5]. This format is based on the ANSI Standard for Interchange of Bibliographic Information [ANSI 4] and was developed in conjunction with efforts of the Inter-Laboratory Working Group for Data Exchange (IWGDE) of the Department of Energy. It provides specifications for:.

1. elemental data types -- numbers and text in code extensions

2. a set of structures -- scalars, vectors and arrays -- with associated format information as well as a higher level hierarchical structure

3. a method of naming or describing the data contained in each field or subfield.

The intent of this proposed standard is to provide the means to interchange a wide variety of information while remaining content-independent. In addition, the proposed standard utilizes the concept of a logical record which is media- independent. The ASCII character set [ANSI 1,2], is recommended as the preferred code for representing all data types, but non-ASCII coded character sets are also permitted.

Partial implementation of this proposed ANSI standard is underway within the IWGDE. By late 1978 it is expected that versions will exist for PL/I (IBM), DEC PDP-11, and a FORTRAN/Assembler (CDC).

3.1.4 Process Interface. Regardless of the implementation approach chosen, run-time support of a remote record access system requires a mutually agreed upon mechanism or protocol for interfacing user/server processes. Such a protocol, termed Interprocess Communication (IPC), provides the basic mechanism for initiating and controlling the flow of data between cooperating processes. Since processes are the only active entities within a computer system, IPC is a basic building block for supporting communication between computers.

Three increasingly sophisticated levels of interprocess communication can be identified: *job level*, *call/return*, and *message based*. At the job level, a basic mechanism is provided for executing a job consisting of a collection of job steps, each of which may be resident on a different system. The IPC mechanism must support initiation of a job step when the required input files are available and must also provide for migration of output files upon termination of the step. Job steps capable of concurrent execution should also be identified. Such a mechanism is provided as part of JES-2 [SIMPR 78] and as part of an Experimental Network Operating System [KIMBS 78].

Job level IPC only supports interaction prior to the initiation or following the termination of a job step. If one wishes to provide a run time mechanism, some attention must be given to the form of implementation. One alternative, the *call/return* based approach, allows one process to communicate with another in a manner directly analogous to subroutine calls. That is, a process issues a CALL and thereafter enters the WAIT state pending RETURN of the results.

Although the *call/return* approach is intui tively straightforward, its use in a networking environment poses certain problems reflecting uncertain delays and the likelihood of outages. In the context of an individual system, aborting a job if a system crash oc curs after a subroutine *call* has been issued is unexceptional. In contrast, in a networked environment, the likelihood of communications network outages or the unavailability of a remote systems can result in exceptionally long processing delays for the *call*ing process. A better approach would be to request initiation of a remote process, continue executing, and later check to see if the desired results have been returned. This constitutes the message based approach to IPC.

Message based IPC provides a very flexible approach for communicating between systems. The cost is the requirement that the user program explicitly provide for transmitting and receiving messages. Although transmission might be considered to be at the same level of difficulty as supporting the *call/return* approach, message reception requires substantially more sophisticated mechanisms. This reflects the desirability of having system support in classifying messages and for permitting inspection of message queues to determine the appropriate sequence for processing. For example, it is usually desirable that a process be immediately notified whenever a remote host is down while, in contrast, handling results returned by a remote process can usually be deferred until a collection of such results are to be processed.

3.1.5 Arithmetic Capability. Representing and manipulating numerical data that exceeds the precision capabilities of the processing host is one of the problems that occur when attempts are made to manipulate data that is in the native form of another processor. It is not acceptable to require that the data "fit" into the word size of the processor supporting data conversion as such a requirement could result in a serious loss of information (e.g., precision loss). Representing such data in character rather than binary form (e.g., character representation of floating point data) would be one approach to the representation problem. Routines are then required that are capable of accepting variable length character (or bit) strings and performing various classes of operations on them (e.g., arithmetic, logical, string, and Boolean).

Although an arbitrary precision arithmetic capability will help prevent data precision loss during the portions of the conversion process of the source record from source host format to canonical NNF format, precision loss may still occur if the word size, for example, of the target host is less than that required to represent the data.

3.2 Architectural Alternatives

As discussed by Shoshani [SHOSA 72,73], there are several possible approaches to data sharing in computer networks in terms of distribution of the support components. Shoshani terms these categories: centralized, standardized, data transformation, and integrated.

3.2.1 Centralized. In the centralized case, network access to a DBMS may involve dealing with a specialized data base machine. Such is the case with the Computer Corporation of America's Datacomputer [MARIT 75]. In this situation programs scattered around the network interact with the Datacomputer in a common Datalanguage. This language includes facilities for describing data, creating and maintaining a data base, and the selective retrieval of items from the data base. Such centralization of DBMS services lifts from the user such tasks as learning more than one query language. However, continuing research in DBMS technology alone is an indication that it is unrealistic to assume that all DBMS-related user needs can be met by a single type of system. Thus, it is reasonable to assume that network users will require access to various DBMSs, and in fact may wish to update the data maintained by one system with information retrieved by another system having perhaps a significantly different architecture.

3.2.2 Standardized. In the standardized approach, the same set of data management services is implemented throughout the network. While this approach might be preferrable under certain circumstances, its implementation on pre-existing systems would be relatively difficult. That there is some movement in this direction is evidenced by the proposed data exchange formats, e.g., [ANSI 5] described above, and current efforts on the part of ANSI, the International Organization for Standardization (ISO) and others in defining a reference model for distributed systems within which standards can be established [ISO 79].

3.2.3 Data Transformation. The data transformation approach involves the reconfiguration of data from the form in which it is maintained on one system, directly into the form required by the system on which data processing is to take place. As Shoshani observed, "the data transformation approach can be viewed as an extension of the centralized approach to handle existing data from existing systems"[SHOSA 72]. Both the DRS and DSCL approaches discussed above are representative of this class.

3.2.4 Integrated. Finally, an integrated approach would involve the use of interfaces and a common language in conjunction with existing data management systems. The interfaces themselves may be physically co-located with the corresponding data management systems, or centrally located at one network location. NBS's XNOS has adopted this type of approach in its support of network data. The XNOS Experimental Network Interface Machines (XNIMs) serve as interfaces between heterogeneous computer and data base systems. A common command language is supported for file maintainence and network job execution [FITZM 78] and XRRA provides the data conversion interface for exchanging structured data. In addition, a Experimental Network Data Manager (XNDM) is now being designed and implemented at NBS to interface heterogeneous DBMSs. Users and processes will express their requests in a standard query language which the XNDM will transform into the DBMS-specific languages [KIMBS 79].

3.3 Design Considerations

Once an architectural approach has been selected, two major alternatives confronting a RRA designer revolve around (i) where to place the RRA support components and (ii) how to interface to other networking capabilities.

3.3.1 Inboarding vs. Outboarding. As illustrated in Figures 3-2a and 3-2b, RRA support components (e.g., translators, data descriptions) may be incorporated inside of existing computer systems (i.e., "inboarding") or special-purpose, perhaps dedicated, front-end or shared systems charged with these responsibilities may be developed (i.e., "outboarding"). The selection of one approach over the other must be based on an analysis of the trade-offs involved in each case.

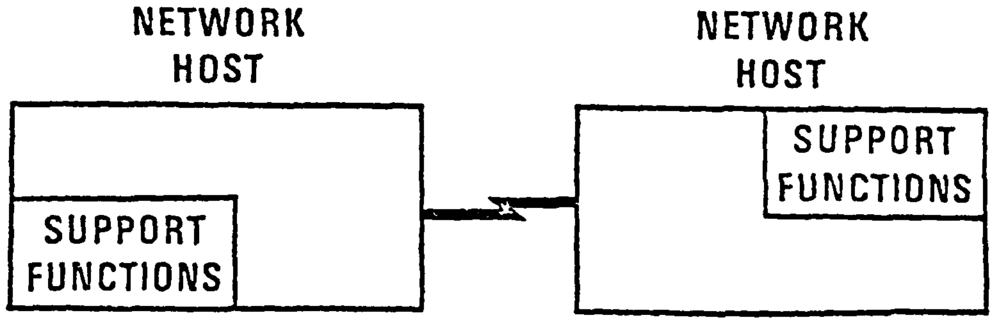


FIGURE 3-2A:
INBOARDING SUPPORT FUNCTIONS

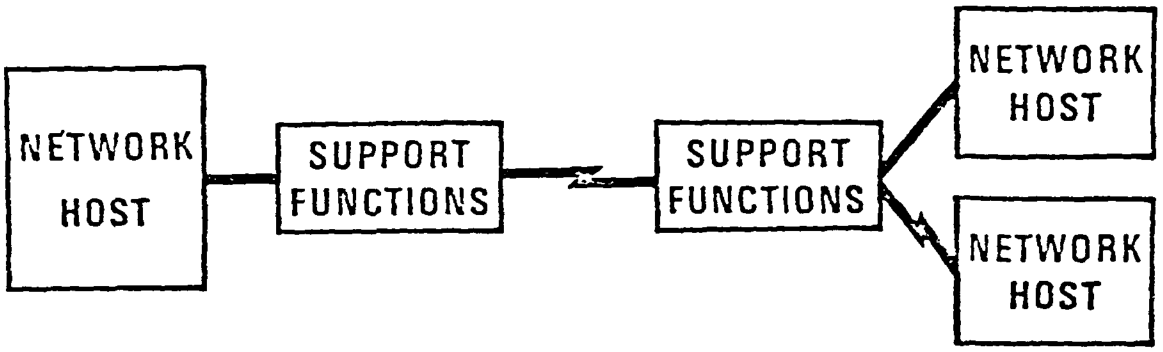


FIGURE 3-2B:
OUTBOARDING SUPPORT FUNCTIONS

The XNOS implementation is an example of "outboarding" as the NOS support functions (including XRRA support) are consolidated into the XNIM. Minimal burden is placed on XNOS-participating hosts.

3.3.2 (De)Centralization. Whether "inboard" or "outboard," RRA support functions may be centralized (i.e., provided by one system) or distributed (i.e., spread across many systems). If "inboarded," then the decision to centralize or distribute these functions would depend on such factors as the overhead involved in implementing a general purpose translator (e.g., Michigan's SDDL) or a set of translators (e.g., DSCL) at a number of hosts. Another factor could be the utility of maintaining a centralized data base manage- ment system which is also capable of translating and transforming records to meet the needs of requesting host systems (e.g., CCA's Datacomputer).

If "outboarding" is chosen, then the demand for support system services would determine the number of systems required. For example, a network supported by XNOS might have one XNIM supporting all participating host systems (e.g., the current XNOS configuration). If demand increased sufficiently, an XNIM might be dedicated to serve one specific class of systems (e.g., Multics systems). In the most distributed case, each participating XNOS host would be served by an XNIM support system.

In the final analysis, the optimal degree of (de)centralization chosen for implementation will depend upon a combination of managerial (e.g., security, control) and physical (e.g., traffic, bandwidth) characteristics.

3.3.3 Layering Concept. Modularity has come to be accepted as the most desirable implementation approach for operating systems and large applications. Anderson et. al. [ANDEA 74] observe that the concept of "layering" is closely related to and in fact includes that of modularity. "Levels" are specified which define precise boundaries between different related sets of modules. At each level, the modules are implemented using the functions provided by lower levels as primitives. These levels are often referred to as "virtual machines" in operating system design.

The following principles have guided the ANSI-ISO effort to design a standard reference model of the architecture of distributed systems [BACHC 78] [DESJR 78]. They are intended for use in determining the number of layers and the best place for boundaries between layers include:

1. Create a sufficient number of layers to divide the total work into pieces small enough for easy comprehension by a single person.

2. Do not create so many layers as to complicate the system engineering task describing and integrating these layers.

3. Create a boundary at a point where the services description can be small and the number of interactions across the boundary are minimized.

4. Create separate layers to handle functions which are manifestly different in the process performed or the technology involved.

5. Collect similar functions into the same layer.

To be consistent with this concept, an RRA capability should be built upon "lower-level" functions that are concerned with transporting data between computer systems. In addition, RRA should be somewhere "above" the layer in which Interprocess Communications functions reside. The exact relationship of RRA to other "higher-level" functions concerned with data exchange on an end-to-end basis (e.g., from operating system to operating system or application process to application process) remains to be fully explored. (See Section 6 for more on this problem.)

4. IMPLEMENTATION APPROACH

In [KIMBS 78], an overall description is given for the implementation of the NBS Experimental Network Operating System. The Experimental Remote Record Access system operates within and as an integral part of the XNOS.

This section describes in more detail the approach adopted in the XRRA implementation. The major components (e.g., functional, informational) are identified, and a detailed example of a session between two processes requiring XRRA services is presented.

4.1 XRRA Architecture

As illustrated in Figure 4-1, the major functional and informational components of XRRA reside on the XNIM. XRRA assumes the existence of a suitable host mechanism for retrieving a record based on utilization of a unique key, if random access techniques are being employed or, alternatively, the keyword 'NEXT' if sequential access is being used. The data conversion approach adopted in XRRA involves the use of non-procedural languages (tables) to implicitly specify the data manipulations.

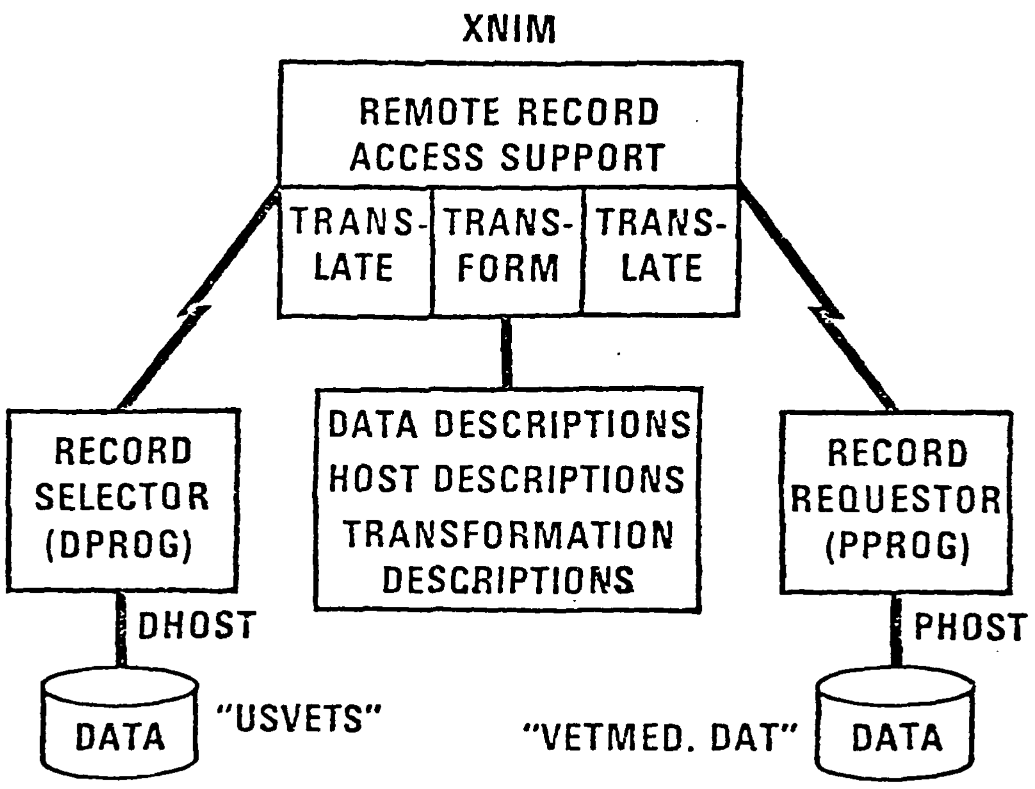


FIGURE 4-1:
XRRA COMPONENTS

The following data types are presently supported: INTEGER, REAL, LOGICAL, BINARY, and CHARACTER.

Conversions of these data types have been successfully performed on all of the systems currently supported by XNOS. Honeywell 6180 running Multics, DECSYSTEM-10 running TOPS-10 and TENEX, and Digital Equipment Corporation 11/45 supporting Bell Laboratories Unix timesharing system. ("Unix" is a Bell System Trade/Service Mark.)

4.2 XRRA Example

The data translation and transformation capabilities supported in providing process access to remote records can best be illustrated by following a record and its associated descriptive tables through the path from the host maintaining the data (DHOST) to the host requesting the data (PHOST). This path is shown in Figure 4-2. The DHOST in this scenario maintains a data base (USVETS) of medical records for veterans. DPROG is the data selector process available on DHOST. The Data Element Descriptions (DED) of the Logical Record Description (LRD) table for these records would then be as shown in Table 4-1.

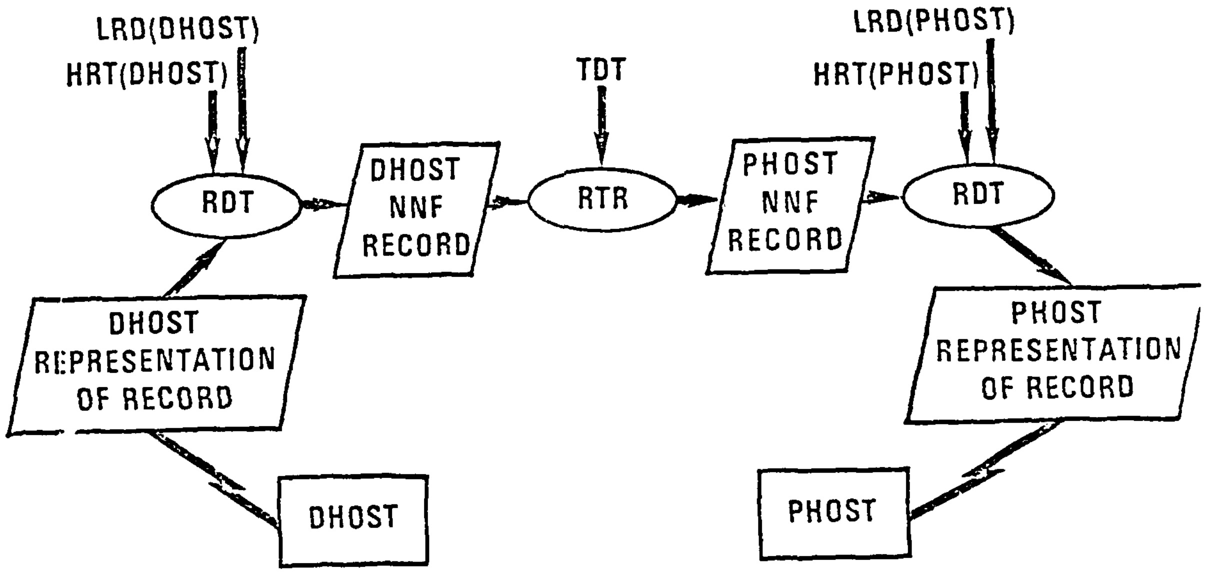


FIGURE 4-2:
DATA TRANSFER PATH

```
ID = 1,1,0,  IDNO, C(9,0)
ID = 1,2,0,  PATIENT, C(20,0)
ID = 1,3,0, BIRTH, C(7,0)
ID = 1,4,0,  ALLERGY, B(36,0)
ID = 1,5,0,  ALLERGY TEST, B(36,0)
ID = 1,6,0, HEIGHT, R(3,2)
ID = 1,7,0,  WEIGHT, I(3,0)
ID = 1,8,0,  SEX, C(1,0)
ID = 1,9,0, DISEASE1, C(0,0)
ID = 1,9,1, DISEASE1, NAME, C(15,0)
ID = 1,9,2, DISEASE1. DATE, C(7,0)
ID = 1,9,3, DISEASE1. MEDICATION, C(15,0)
ID = 1,10,0,  DISEASE2, C(10,0)
ID = 1,10,1,  DISEASE2. NAME, C(15,0)
ID = 1,10,2,  DISEASE2. DATE, C(7,0)
ID = 1,10,3,  DIRSEASE2. MEDICATION, C(15,0)
ID = 1,11,0,   DISEASE3, C(0,0)
ID = 1,11,1,   DISEASE3. NAME, C(15,0)
ID = 1,11,2,   DISEASE3. DATE, C(7,0)
ID = 1,11,3,   DISEASE3. MEDICATION, C(15,0)
ID = 1,12,0,  DISEASE4, C(0,0)
ID = 1,12,1,  DISEASE4. NAME, C(15,0)
ID = 1,12,3,  DISEASE4. DATE, C(7,0)
ID = 1,12,3,  DISEASE4. MEDICATION, C(15,0)
ID = 1,13,0,  PARENTS, C(0,0)
ID = 1,13,1,  PARENTS. MOTHER, C(20,0)
ID = 1,13,2,  PARENTS. FATHER, C(20,0)
ID = 1,14,0,  YRS CIVILIAN GOVT, I(2,0)
ID = 1,15,0,  YRS MILITARY, I(2,0)
ID = 1,16,0,  DISABLED VET, L(1,0)
```

TABLE 4-1:
DED FOR "USVETS" RECORD

PPROG is a process executing on PHOST which requires access to data that is available on DHOST. The PHOST data requirements constitute a subset of the DHOST data record. Thus, certain data fields (e.g., parent information) are not needed by the PHOST process, and in fact should not be transmitted. In addition, suppose that several other fields are to be added, or otherwise operated upon. The result of these operations would be a somewhat smaller, but in any case transformed, PHOST record as defined by the LRD shown in Table 4-2 and maintained on the XNIM.

```
ID = 1,1,0, SSN, C(9,0)
ID = 1,2,0, NAME, C(20,0)
ID = 1,3,0, SEX, C(1,0)
ID = 1,4,0, YRS GOVT SERVICE, I(2,0)
ID = 1,5,0, BIRTH DATE, C(7,0)
ID = 1,6,0, TESTED ALLERGIES, B(36,0)
ID = 1,7,0, SUSPECTED ALLERGIES, B(36,0)
ID = 1,8,0, WT, I(3,0)
ID = 1,9,0, HT, R(3,2)
ID = 1,10,0, DISEASE 1, C(37,0)
ID = 1,11,0, DISEASE 2, C(37,0)
ID = 1,12,0, DISABLED, L(1,0)
```

TABLE 4-2:

DED FOR "VETMED.DAT" RECORD

For each DHOST and PHOST logical record type, a Transformation Description Table (TDT) is then provided. This table, shown in Table 4-3, establishes the relationships between data items in the PHOST and DHOST records using data element names from the DED portion of the Logical Record Description Table and the operators specified in Table 3-1.

PHOST RECORD	DHOST RECORD
SSN	= IDNO
NAME	= PATIENT
SEX	= SEX
YRS-GOVT-SERVICE	= YRS-CIVILIAN-GOVT & YRS-MILITARY
BIRTH-DATE	= BIRTH
TESTED-ALLERGIES	= ALLERGY & ALLERGY-TEST
SUSPECTED-ALLERGIES	= ALLERY | ALLERGY.TEST
WT	= 2.2 × WEIGHT
HT	= (0.4 × HEIGHT)/12
DISEASE-1	= DISEASE3.NAME #DISEASE3.MEDICATION #DISEASE3.DATE
DISEASE-2	= DISEASE4.NAME #DISEASE4.MEDICATION #DISEASE4.DATE

TABLE 4-3:
EXAMPLE TRANSFORMATION DESCRIPTION TABLE

Several interesting capabilities, which result from supporting tree-structured data records, are illustrated in this example. Note that when selecting the last two disease history fields (DISEASE3 and DISEASE4) from the DHOST record for inclusion in the PHOST record, the fields, which are each composed of three subfields, are transformed via reordering and concatentation of the related subfields. The result is then assigned to the appropriate field in the PHOST record description. For example, the TDT contains the entry

"DISEASE-1 = DISEASE3.NAME # DISEASE3.MEDICATION # DISEASE3.DATE"

where '#' is the concatenation operator. This statement is functionally equivalent to the following set of statements:

DISEASE-1.NAME = DISEASE3.NAME

DISEASE-1.TREATMENT = DISEASE3.MEDICATION

DISEASE-1.OCCURRENCE = DISEASE3.DATE

Thus, one entry in the TDT describes a 3-part PHOST disease history field.

For every supported host type, necessary host-descriptive information is in the Host Representation Table (HRT) (e.g., Table 3-2). For this example, XRRA would require HRTs for the Honeywell H6180 and DEC PDP 11/45. In the initial XRRA implementation it is assumed that all data types are single precision.

The following sequence of events occur during an XRRA session:

1. A user requests activation of the PHOST process, PPROG. A user requests activation of the PHOST process, PPROG.

2. The XNIM activates the DHOST process, DPROG.

3. PPROG requests for data are intercepted and passed on to the awaiting DPROG.

4. DPROG retrieves the indicated data and returns it, in native form (i.e., binary strings) to the XNIM.

5. The XNIM then directs this data string to the Record Translation/Transformation (RTT) component of XRRA, identifying (via calling parameters) the DHOST and PHOST names, along with the LRDs which describe the data.

6. The Record Data Translation component of RTT translates the DHOST record to Network Normal Form.

7. The DHOST record in NNF is then transformed by the Record TRansformation component of RTT to meet the PHOST format requirements, as indicated by the Tranformation Record Table.

8. The resulting PHOST record in NNF is translated into PHOST native representation.

9. The PHOST record is returned to the XNOS monitor in the XNIM, which then transmits the record to the awaiting PHOST process, PPROG.

The record received from the DHOST at step 4, appears to the XNIM as a seemingly meaningless binary string represented by the character stream shown in Figure 4-3a. Figure 4-3b, however, shows the DHOST record as it would appear within the XNIM after being translated to Network Normal Form (NNF) by the Record Data Translator (RDT) portion of the Record Translator/Transformer (RTT) in step 6. Notice that trailing blanks have been dropped in CHARACTER data items, INTEGER and REAL data elements have been given explicit signs as well as decimal points, as appropriate, and BINARY fields have been "exploded" into character strings. In addition, the BOOLEAN field, DISABLED-VET, is now represented by the string '"T"'.

```
BKINEGKDGBLANMGODHBLIIAEACACBJKAMCHCDGBJEOGFPCMBHMJEGPDHBJEOGCABAA
IAEACABIIMAGEDGBMIMIGACAAPAPAPAPAPAPAPAPAPAPAIELCAAAAAAAAAAAADCCGIIAE
ACADGJIENIGBDJBKEMCCABAAIAEACABAAIAEACABIAMEGADBBMINAGACADDJKENMFP
DAJLIMIFPDKBLMNMGJDBIIAEACADKBJENMGEDHJLINCHEDEJMMEACA
BAAIAEACABIAMIGADCBMINEGACADBJLMOEHEDEJMMNOGODCIIAEACABAAIAEACADJJ
NAOEGFDIAIAEACABAAIAEACABAAIAEACABIAMMGADDBMINIGACADIBJENMGJDBJKEN
IGMDEJLIEACABAAIAEACADDBLAOKCABAAIAEACABAAIAEACABAAIAEACABIANAGADE
BMINMGACADJBJEOGHEBAAIAEACABAAIAEACABAAIAEACACJJNEOGGB
DHBHMIEFPCAJLIOIGIDHJLIPCCABAAIAEACACBJKAMCHCDGBJEOGFPCFBLMNMGFDJI
IAEACABAAIAEACAAAAAAAABEAAAAAAAABEAAAAAAAAA
<O>
```

FIGURE4-3a: XNIM-BASED CHARACTER REPRESENTATION OF DHOST RECORD

```
555667777\Charles X Jones      \1026920\+000111100001111000001
1110000111100000\-111000011110000111100001111000011111\+150.25
\+50\M\malaria      \0101940\gin and tonic   \tendonitis
  \0202950\cortisone     \strep     \0303960\penicillin
    \flu      \0404970\rest       \Susan B Anthony
    \Charles Jones        \+20\+20\"F"\
```

FIGURE 4-3b: DHOST RECORD IN NNF

Once the record is in this form, it is ready for input to the record transformation routine (RTR) at step 7. RDT handles all translation to and from Network Normal Form, while RTR performs the required operations (e.g., +) on the data items, with the results shown in Figure 4-4a. This is actually the Network Normal Form of the record as expected by the PHOST. Note that all parent-related data has been dropped in this example, as no entry referring to those fields exists in the Transformation Description Table.

```
555667777\Charles X Jones    \M\40\1026920\ + 00000000 0000
000000000000000000000000\-111111111111111111111111111111111\11
0.0\5\strep   penicillin   0303960\flu   rest   0404970\*F*\
```

FIGURE 4-4a: · PHOST RECORD IN NNF

Finally, this version of the record must be passed through the RDT at step 8 in order to produce the record in the native, PHOST format, required by PPROG. The resulting translated, transformed, and translated record, Figure 4-4b, is transmitted on to the awaiting PHOST process PPROG.

```
DFDFDGDFDHDGDHDHAADHGIEDHCGBGFGMFPHDFPFIGPEKGFGOCAHDCACACACAAAENAA
CIDADBDGDCDCDJAADAAAAAAAAAAAAAAAEBKAAAAAHEHDGFHCCAHACACACACACACACACA
HACAGOGFGDGJGMGJGJGMCAGOCACACACADDDADDDADGDJAADAGMGGCAHFCACACACACA
CACACACACAHCCAHDGFCAHECACACACACACACACACACADEDADEDADHDJAADAAAAA
```

FIGURE 4-4b: XNIM-BASED CHARACTER REPRESENTATION OF PHOST RECORD

In this example, PPROG then displays the record as shown in Figure 4-5.

```
(phost)Output is:
ssn[c9] = 555667777
name[c20] = Charles X Jones
sex[c1] = M
yrs g[i2] = 40
birth[c7] = 1026920
teste[b36] = 0
suspe[b36] = 0
wt[i3] = 0
ht[r3] = 5.0000000
dis1[c37] = strep   penicillin   0303960
dis2[c37] = flu   rest   0404970
```

FIGURE 4-5: DISPLAY OF DATA AT PHOST

5. PERFORMANCE CONSIDERATIONS

Once the feasibility of a concept has been demonstrat- ed, questions naturally arise regarding the practicality of the approach. Practicality devolves into two issues: i) feasibility of implementation, and ii) performance of the result. Based upon our implementation of an RRA mechanism within the NBS Experimental Computing Facility, we have no reason to believe that the construction of a production mechanism for supporting the meaning of data being transmitted between heterogeneous systems poses any major problems. Thus, it is appropriate to consider the performance issue.

To estimate RRA bandwidth, we will assume that both request and response packets are approximately 1000 bits in length, that both request and response travel through two intermediate packet switches and that the average distance between packet switches is 500 miles. Using 100 000 miles per second as the speed of electric flow in copper wires, it follows that the average time for a packet to move between packet switches is 5 ms. Moreover, assuming 50 Kb. lines, the average time to encode a packet is 20 ms. A total of three encodings are required (source, and two intermediate nodes). Thus, the average time for a packet to travel from source to destination is 75 ms. excluding processing and queuing times. It follows that the round trip time is 150 ms. It follows that even if the remote data could be instantaneously transferred into a buffer, the maximum processing rate would be approximately 6.6Kb/second and the bandwidth against the DBMS would be approximately 6.6 Kb.. Since accessing data in remote systems is likely to require a significant amount of time, the actual bandwidth is likely to be significantly lower, perhaps on the order of 1-2 Kb.

The preceding result is of more than passing interest. To provide an appropriate context, we observe in accordance with Scott-Morton [LUCAH 76] that information processing can be divided into three major categories: operational control, managerial control, and strategic planning. As one passes from operational control to strategic planning both the bandwidth and the predictability of the requirement decrease. Thus, operational control applications are typically high bandwidth and very predictable, e.g. payroll. In contrast, strategic planning requirements are intrinsically low bandwidth and very unpredictable, e.g. which ships are close to a country undergoing a revolution.

Given this context, we are led to conclude that remote access to data in support of operational control is likely to be unsatisfactory. In support of managerial control, it may be unsatisfactory, and in support of strategic planning it is likely to be very satisfactory. As a close corollary, a generalized principle of locality applies. Conceptually, this principle states that: "remote data should be rarely accessed."

In considering these somewhat philosophical comments, it is important to bear in mind that they are predicatd on existing communications technology, e.g. relatively low bandwidth, relatively high cost communications based on using circuits provided by common carriers. Satellite transmission promises a much higher bandwidth at a much lower cost. Nevertheless, in view of transmission delays (.5 seconds round trip), it is still unlikely that high bandwidth remote applications can be effectively supported unless there is a very substantial predictability in the data to be accessed. That is, applications in which large amounts of data can be prespecified are likely to prove more appropriate than those for which it is infeasible to predict future data requirements.

6. STRUCTURED DATA TRANSFER PROTOCOLS

If the requirements for a RRA capability are examined from a more general perspective, insight is gained regarding the specification of basic protocols supporting the exchange of structured data.

Such a Structured Data Transfer Protocol (SDTP) may be viewed as a mechanism which facilitates the sharing of structured data between processes in a computer networking environment. Such exchange of structured data between processes mandates a means of specifying and executing a transformation between different physical and organizational data formats and representations. Specification, creation and/or selection of records to be exchanged via a SDTP would be included in the set of responsibilities of the processes invoking the SDTP.

A specification for a SDTP would consist of the following:

1. a standard format for the exchange of structured data.

2. the information required to describe the exchanged data.

3. the control information (i.e., commands) needed to signal the establishment, maintainence, operation and termination of a connection between SDTP processes.

4. flow and error control responsibilities.

A SDTP would assume the existance of lower level services which would provide the means for reliable transport of information between specified, cooperating processes on a network. It should support interactive use by both humans and processes. Consequently, its operation must be completely deterministic.

Existing standards could prove useful in the development of a standard SDTP. Among these are the ANSI Code for Information Interchange (ASCII) [ANSI 1,2], the standard for character representation of numeric values [ANSI 3], the standard format for exchanging bibliographic information on magnetic tape [ANSI 4], and the proposed standard for data descriptive files [ANSI 5].

In conjunction with the selection of standard formats, an assessment should be made of current and projected requirements for structured data interchange. For example, the cost benefits of providing SDTP support for only character-encoded, structured data should be considered, vs. those for full support of other data types (e.g., binary, real). In addition, the cost vs. benefits of developing and using a SDTP supporting self-describing data (e.g., [ANSI 5]) vs. the transmission of data independently of descriptive information (e.g., XRRA approach) should be evaluated.

7. FUTURE WORK

Remote Record Access is a prime component of general purpose network operating systems. The design and implementation of the described capability has provided a wealth of information about the capabilities and limitations of various approaches to exchanging structured data. This knowledge in turn can prove useful in the development of specifications for (much needed) Structured Data Transfer Protocols. Higher level data sharing services, such as those supporting structured file transfer and distributed data base management, may in turn be built upon such a foundation.

Already the widespread interest in and use of Data Base Management Systems is stimulating investigations into the implications of marrying computer networking and DBMS technology [BOOTG 72,76] [BERGJ 76] [KIMBS 79]. Furthermore, with the rapidly growing dependence on computer networking technology to meet information management and communications needs in government and industry, it is clear that the time is "ripe" for deveopment of standardized, high level communications protocols including those for structured data transfer. Around the world, efforts are underway to develop standards which will facilitate the use of computer networking technology (e.g., ANSI, ISO, CCITT). At the National Bureau of Standards, the development of high level computer networking protocols is a part of a larger effort geared towards the development of an entire "family" of computer system and network standards. These are intended to permit the successful interconnection of competitively procured computer system and network components. Through the development and use of such standards it is believed that the performance and cost advantages of competitively procured systems and components can be used to full advantage by Federal agencies, while at the same time assuring reliable and efficient system operation.

REFERENCES

[ANDEA 71] Anderson, A., et. al., The Data Reconfiguration Service - An Experiment in Adaptable Process/Process Communication, Proceedings of the Second Symposium on the Optimization of Data Communication Systems, IEEE Press, October 1971.

[ANDEA 74] Anderson, A.K., J.W. Benoit, M.A. Padlipsky, Description of the Prototype WWMCCS Intercomputer Network (PWIN) Protocols, Mitre Corporation, MTR-5444, December 1974.

[ANSI 1] Code Extension Techniques for Use with the 7-Bit Coded Character Set for American National Standard Code for Information Interchange (FIPS 35), American National Standards Institute, X3.41-1974.

[ANSI 2] Code for Information Interchange, American National Standards Institute, X3.4-1977.

[ANSI 3] Representation of Numeric Values in Character Strings for Information Interchange, American National Standards Institute, X3.42-1975.

[ANSI 4] Standard for Bibliographic Information Interchange on Magnetic Tape, American National Standards Institute, Z39.2-1971.

[ANSI 5] American National Standard Specification for an Information Interchange Data Descriptive File, American National Standards Institute, X3L5-1978.

[BACHM 79] Bach, M. J., N. H. Goguen, M. M. Kaplan, "The Adapt Data Translation System and Applications," Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks, August 1979.

[BERGJ 76] Berg, John L. (ed.), Data Base Directions: The Next Steps, (proceedings of the Workshop of the National Bureau of Standards and Association for Computing Machinery, October, 1975), National Bureau of Standards, Special Publication 451, September 1976.

[BIRSE 76] Birss, Edward W., and James P. Fry, "Generalized Software for Translating Data," Proceedings of 1976 National Computer Conference, AFIPS Press, Montvale, New Jersey, 1976, pp. 889-897.

[BOOTG 72] Booth, Grayce M., "The Use of Distributed Data Bases in Information Networks," Computer Communication: Impacts and Implications, First International Conference on Computer Communications (October 1972), pp. 371-376.

[BOOTG 76] Booth, Grayce M., "Distributed Information Systems," Proceedings 1976 National Computer Conference, AFIPS Press, Vol. 45, pp. 789-794.

[CERFV 72] Cerf, Vinton G., Eric Harslem, John Heafner, Robert Metcalfe, and James White, "An Experimental Service for Adaptable Data Reconfiguration," IEEE Transactions on Communications, 20:3, (June 1972), pp. 557-564.

[CROCS 72] Crocker, S.D., J.H. Heafner, R.M. Metcalfe, and J.B. Postel, "Function Oriented Protocols for the ARPA Network," Proceedings of the Spring Joint Computer Conference, AFIPS Press, Vol. 40, 1972, pp. 271-279.

[FITZM 78] Fitzgerald, M.L., Common Command Language for File Manipulation and Network Job Execution: An Example, National Bureau of Standards, Special Publication 500-37, August 1978.

[FOLTH 78] Folts, H.C., "Evolution Toward a Universal Interface for Data Communications," Proceedings of International Conference on Computer Communications, 1978.

[FORSH 77] Forsdick, H.C., R.E. Schantz, and R.H. Thomas, "Operating Systems for Computer Networks," BBN Report No. 3614, Bolt Beranek and Newman, Cambridge, Mass., 1977.

[FRYJ 72A] Fry, James P., Diane P. Smith, and Robert W. Taylor, "An Approach to Stored Data Definition and Translation," Stored Data Definition and Translation (SDDT) Task Group, Proceedings of 1972 ACM SIGFIDET Workshop: Data Description, Access and Control, A.L. Dean (ed.), Association for Computing Machinery, 1972

[FRYJ 72B] Fry, James P., Randall L. Frank, and Ernest A. Hershey III, "A Developmental Model for Data Translation," Proceedings of 1972 ACM SIGFIDET Workshop: Data Description, Access and Control, A.L. Dean (ed.), Association for Computing Machinery, 1972.

[FRYJ 74] Fry, James P., and David W. Jeris, "Towards a Formulation and Definition of Data Reorganization," Proceedings of 1974 ACM SIGMOD Workshop on Data Description, Access and Control, Randall Rustin (ed.), Association for Computing Machinery, 1974, pp. 83-100.

[FRYJ 78] Fry, James, Internal report prepared for National Bureau of Standards, 1978.

[HARSE 71] E.F. Harslem and J.F. Heafner, The Data Reconfiguration Service - An Experiment in Adaptable, Process/Process Communication, RAND Corporation, R-860-ARPA, November 1971.

[HARSE 72] Harslem, E., and J. Heafner, The Data Reconfiguration Service Compiler: Communication Among Heterogeneous Computer Centers Using Remote Resource Sharing, RAND Corporation Technical Report R-887-ARPA, April, 1972.

[HOUSB 77] Housel, B.C., N.C. Shu, R.W. Taylor, S.P. Ghosh, and V.Y. Lum, EXPRESS: A Data Extraction, Processing, and Restructuring System, IBM Research Laboratory, RJ1962(27742), 1977.

[ISO 79] ISO/TC97/SC16 N227, "Reference Model of Open Systems Interconnection," August 1979.

[KIMBS 76] Kimbleton, S.R. and R.L. Mandell, "A Perspective on Network Operating Systems," Proceedings 1976 National Computer Conference, AFIPS Press, Montvale, N.J., Vol. 45, 1976, pp. 551-559.

[KIMBS 77] Kimbleton, S.R., and R.L. Mandell, "A Perspective on Network Operating Systems," Proceedings of 1976 National Computer Conference, AFIPS Press, Vol. 45, 1976, pp. 551-559.

[KIMBS 78] Kimbleton, Stephen R., Helen M. Wood, and M.L. Fitzgerald, "Network Operating Systems -- An Implementation Approach," Proceedings 1978 National Computer Conference, AFIPS Press, Montvale, N.J. Vol. 47, 1978, pp. 773-782.

[KIMBS 79] Kimbleton, Stephen R., Pearl S.-C. Wang, and Elizabeth N. Fong, "XNDM: An Experimental Network Data Manager," Proceedings of the Third Berkely Workshop on Distributed Data Processing, 1979.

[LEVIP 77] Levine, Paul H., Facilitating Interprocess Communication in a Heterogeneous Network Environment, MIT/Laboratory for Computer Science, MIT/LCS/TR-184, July 1977.

[MARIT 75] Marill, Thomas, and Dale Stern, "The Datacomputer - A Network Data Utility," Proceedings of the National Computer Conference, 1975, AFIPS Press, pp. 389-395.

[MERTA 74] Merten, Alan G., and James P. Fry, "A Data Description Language Approach to File Translation," Proceedings of 1974 ACM SIGMOD Workshop on Data Description, Access and Control, Randall Rustin (ed.), Association for Computing Machinery, 1974, pp. 191-205.

[ROTHJ 77] Rothnie, J.B., and N. Goodman, An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases, Computer Corporation of America, Technical Report CCA-77-04, March 1977.

[SCHNG 75A] Schneider, G. Michael, "DSCL - A Data Specification and Conversion Language for Networks," Proceedings of ACM SIGMOD Conference, May 1975, Association for Computing Machinery, 1975, pp. 139-148.

[SCHNG 75B] Schneider, G. Michael, and E.J. Desautels, "Creation of a File Translation Language for Networks," Information Systems, 1:1, (January 1975), pp. 23-31.

[SHOSA 72] Shoshani, Arie, "Data Sharing in Computer Networks," Proceedins of 1972 Wescon Electronic Show and Convention, September 1972, Institute of Electrical and Electronics Engineers.

[SHOSA 73] Shoshani, A., and I. Spiegler, "The Integration of Data Management Systems on a Computer Network," Proceedings of AIAA Computer Network Systems Conference, American Institute of Aeronautics and Astronautics, 1973, AIAA Paper No. 73-417.

[SHUN 76] Shu, N.C., V.Y. Lum, B.C. Housel, An Approach to Data Migration in Computer Networks, IBM Research Laboratory, 1976.

[SHUN 77] Shu, N. C., B. C., Housel, R. W. Taylor, S. P. Ghosh and V. Y. Lum, "EXPRESS: A Data EXtraction, Processing, and REstructuring System," ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977.

[SIMPR 78] Simpson, R. O., G. H. Phillips, "Network Job Entry Facility for JES2", IBM Systems Journal, International Business Machines Corporation, Vol. 17, No. 3, 1978.

[SIPPC 72] Sippl, Charles J., and Charles P. Sippl, Computer Dictionary and Handbook, Howard W. Sams and Co., Inc., 1972.

[SMITD 72] Smith, Diane C.P., "A Method for Data Translation Using the Stored-Data Definition and Translation Task Group Languages," Proceedings of 1972 ACM SIGFIDET Workshop: Data Description, Access and Control, A.L. Dean (ed.), Association for Computing Machinery, 1972.

[TREMJ 76] Tremblay, J. P., An Introduction to Data Structures with Applications, McGraw-Hill, Inc. 1976.

APPENDIX B

THE DESIGN OF A DATABASE ACCESS PROTOCOL

Pearl S-C. Wang

Stephen R. Kimbleton

National Bureau of Standards

June 1980

ABSTRACT

This paper describes the development of a high-level communications protocol for network-wide database access. Design decisions concerning the range of applicability, the ease of use, the ease of implementation and the performance of the protocol are discussed. The protocol is based upon a user-server model of asymmetric interactions and a canonical database management system--the Network Virtual Data Manager. A prototype implementation is currently being undertaken at the National Bureau of Standards. Early experiences concerning the protocol and its implementation are also briefly described.

1. INTRODUCTION

The development of computer-communication networks facilitates the sharing of a wide range of computing resources such as hardware, programs and data. A common problem in providing the capability for users to use remote resources is the need for establishing a set of conventions to allow meaningful communication between processes running on dissimilar hosts (Crocker 72). This paper describes the design of one such set of conventions: a database access protocol (DAP) to allow network users easy access to the databases managed by remote database management systems (DBMSs) and the data retrieval, manipulation and control facilities these systems provide (Date 77). The present work focuses on the various issues, objectives and constraints that led to the final design. A prototype implementation is currently underway and the experiences we derived from it have been reported elsewhere (Kimbleton 79), (Kimbleton 80).

The purpose of the protocol is to promote sharing of databases by providing methods for the convenient use of existing remote DBMS capabilities. It is built to achieve the principal aims of: generality, ease of use, ease of implementation and efficiency. These four points are now dealt with in turn:

i) The protocol is designed to accommodate as many different types of invocation mechanism and as wide a range of remote machines and DBMSs as possible. It can be used both by humans and by programs acting on their behalf. Although DAP has a strong bias towards relational-calculus (Codd 71) systems (this point will be made clear by discussions in Sec. 4), prototype implementations have been carried out for both relational-calculus and Codasyl DBTG systems (Codasyl 73) and no particular problems related to the design of the protocol have been encountered in handling either type.

ii) Considerations of usability divide into three parts. In terms of execution control, CAP provides functions for a user to obtain information about, and to control the progress of request processing at remote hosts. In terms of command structure, it is a high-level, non-procedural, table-based query language derived from SQL (Chamberlin 76), a human-engineered data manipulation language. In terms of communications control, DAP handles all the necessary opening/closing of connections on demand. Thus, to the user, DAP is a connectionless protocol--the user maintains no information on the communication state.

iii) The DAP protocol is built upon the idea of a network virtual date manager and a user-server model of asymmetric interactions (See Figure 1 and further discussions in Sec. 3). This reduces the problems of handling the diversity of remote DBMSs to a manageable size. It also provides an implementation framework that is ideally suited to the functional partitioning of protocol modules into common and target-unique tasks. A second contributing factor to the ease of implementation is the decision to have protocol transactions share common boundaries with database transactions. DAP is therefore insensitive to the history of query processing at the serving database systems. As a result, the number of distinct states of the protocol is reduced significantly.

Given the general requirement to support a functionally complete set of database query capabilities, we tried to keep the protocol as small as possible. Although the set of DBMS facilities included is probably not "minimal", we believe it provides a basic set that is both necessary and adequate. Aside from a few intentionally incorporated features, DAP contains no redundancies (i.e. equivalent ways for requesting the same service).

iv) The performance of DAP has the following dual aspect:

a) Its transaction orientation minimizes the number of message exchanges among remote hosts and as a consequence, allows effective utilization of the underlying communications subnet (Gray 79).

b) Its relational-calculus query basis provides a degree of data independence so that significant speedup of the system is possible. Typical of these potential improvements are target-query optimization (Smith 751), (Selinger 791), and optimization with respect to distribution effects (Wong 77). (Hevner 79). The implementation of these speedups is further aided by the separation and assignment of target-DBMS specific functions to individual servers, allowing custom-tailoring of these modules to take advantage of individual DBMS characteristics in formulating optimization strategies.

Because the diversity and the complexity of DBMSs play a large role in the design of database access protocols, the next section (Sec.2) examines in detail a number of issues relating to it. We then present the solution approach adopted for DAP and discuss the underlying rationale. In the section that follows (Sec. 3), we turn our attention to issues relating to the distributed nature of the implementation environment --to considering what it means when the DAP functions has to be performed by multiple, interconnected computers and to considering how we chose to partition and assign the pieces of software to different processors. Section 4 provides a complete description of the DAP protocol in terms of the syntax and semantics of the individual commands. Brief discussions of the pragmatics of the protocol's implementation and use are also included. Section 5 contains a summary of the work, indicating what it includes and what it deliberately excludes. The final section (Sec. 6) discusses the current status of our implementation effort with some concluding remarks.

2. THE NETWORK VIRTUAL DATA MANAGER

Even though all generalized database management systems perform essentially identical functions, they differ widely both in the data models they employ and in the data languages they provide for interacting with the data model (Date 77). To support the many different types of remote DBMSs, we have chosen to define a canonical database system: the network virtual database manager (NVDM) (1). All database requests that produce the same result are equivalent to an NVDM request. A user is free to issue queries expressed in non-canonical forms, and these queries are automatically translated into the common, network virtual form. A further translation then maps the NVDM query into its target DBMS form before sending it off to the remote system. (We note here that our focus so far have been on the management of the heterogeneity problem of the target DBMSs. Issues related to the accommodation of different types of user queries have only received cursory attention and no such translation capability has been implemented yet.)

In order to model the wide variety of DBMSs currently available, the NVDM has to be high level both in its data structuring rules, and in its queries and data manipulation commands since we rely on the remote DBMSs for all database management services and some remote DBMSs only allow high-level, non-procedural data accesses (Codd 70).

The NVDM consists of four closely interrelated but nonetheless separate pieces: a Network Virtual Scheme (NVS) and its associated Network Virtual Data Definition Language (NVDDL) to define a set of acceptable data structures, a Network Virtual Query Language (NVQL) for formulating data retrieval requests, a Network Virtual Update Language (NVUL) for database updating, and a Network Virtual Data Control Language (NVDCL) for ensuring data integrity and for enforcing access controls. The current paper only considers the design of the first two components. The reader is referred to (Fong 80) and (Kimbleton 79) for discussions of the other aspects of the system.

2.1 The Network Virtual Schema

The selection of an appropriate logical data structure class, that is, a network virtual data model, should be based on user capabilities and needs. For network users it is unrealistic to assume extensive knowledge of remote

(1) A word of precedence might be appropriate here. We did not invent this term. As far as we are aware, it was used first by (Gardarin 77) and also by (Gray 79) in this context.

database organizations. This, coupled with considerations of implementation issues in mapping between a users' or a remote DBMSs' view of the database and the NV3, argues for a simple and representationally clear data model. The virtual view of data which the NVDM presents has further relevant implications. The first is the effective elimination of the utility of performance tuning based on the anticipated nature of the user's requests The second is that system maintained record-type interrelationships--such as those provided by Codesyl DSTG and hierarchies (Date 77)-- are neither feasible nor useful.

These observations suggest using a relational data model as our basic point of departure. The relational model does suffer from a number of defects and limitations, however (Deheneffe 74), (Hainaut 74), (Chen 76) The most serious is its inadequacies in capturing more of the meaning of the data. Fortunately, for an NVS, the virtual view permits incorporation of additional data semantics with minimal effort, and this can be done entirely independently from the NVS design process. We therefore decided to proceed with a design based on the relational model.

2.2 The Network Virtual Query Language

Given a relational date model, the basic criterion we have tried to adhere to in choosing a query language is simplicity. We want to use our network virtual query language to the extendible to future NVQLS. Therefore it has to be simple and well-understood--by users as well as implementers This argues for a "minimal" language, that is, one which contains only those features that are necessary for basic data retrieval over all DBMSs. However, there are two other important constraints: the representativeness of the language and its usability. The first constraint is due to our desire to demonstrate our ability at handling the typical set of database query facilities. Also, in order not to become a mere "toy" exercise, the language should have enough expressive power to enable a user to make the best use of the facilities provided by the host DBMSs.

The above considerations (and our unwillingness to introduce yet another query language) led to our decision to work within the framework of SQL (Chamberlin 76), a table-based data language for IDM's System R (Astrahen 76). However, as the reader will see from later discussions, the current version of NVQL (2) departs from its parent language in a number of respects

Many SQL constructs which are redundant (e.g. the set comparison operator "in") or unnecessary (for database retrieval purposes, e.g. the "order by" clause) have been omitted. As a remark on the elimination process, we note

(2) This was called XNQL in our previous work (Kimbleton 79), Kimbleton 80).

that due care has been taken to retain a relationally complete subset (Codd 71). (A demonstration of this fact is given in Table 1.) Other features, for example, those needed for specifying the target databases. have been added. A detailed analysis of our preliminary design is given in the next section. Although it is still too early to judge the success or failure of the design, our initial experience (Kimbleton 80) suggest that the query language is in fact easy-to-use, easy-to-implement and capable of supporting widely differing types of DBMSs.

So far we have examined the design issues arising from the heterogeneous nature of the individual DBMSs. Our next section will examine the second fundamental aspect of the problem: the distribution of the implementation environment and its role in shaping the design of DAP.

3. THE DAP PROTOCOL MODEL

A complete description of a protocol should specify: i) the set of services provided to the users, ii) the specific means for interfacing with the users, and iii) the internal structure of the "protocol machine" (Sunshine 79). In a distributed environment, the protocol machine must itself be implemented in a distributed fashion, with communicating "stations" local to each of the participating hosts. The interactions among these stations define the actual structure of the protocol machine.

Although the first two components are essential in making a formal specification complete, we shall, in the present paper, (as we have in our work), only look at the actual DAP--that is, the laws governing the functioning of the different protocol stations. This is primarily because we are basically interested in the internal workings of protocols. Also, we feel that the service and interface specifications should be "frozen" only when it is necessary--after we have had some operational experience with our prototype implementation.

3.1 Partition of DAP Functions

With the above preamble, let us now describe the structure of the protocol, DAP is organized into a collection of independent, concurrent, cooperating processes or stations. Given this general approach, the question arises as to how to partition the various protocol functions among the different stations. This functional partitioning is probably the single most critical issue facing the designer of a database access protocol. The failure to arrive at a proper "separation of concerns" will undoubtably led to a poor design. In particular, the result will certainly be difficult to implement and impossible to modify or extend, as has been pointed out by many software designers and implementers (Parnas 72), (Dijkstra 76).

Table 1. The Relational Completeness of NVQL

Relational Algebra	NVQL
R ∪ S	select x.* from R x union select y.* from S y
R ∩ S	select x.* from R x intersect select y.* from S y
R ÷ S	select x.* from R x minus select y.* from S y
R ⊖ S	select x.*, y.* from R x, S y
R[r1,r2,...]	select x.r1, x.r2, ... from R x
R[r1<comp←op>s1]S	select x.r1, x.r2, ... ,y.s2, ... from R x, S y where x.r1 <comp←op> y.s1
R[{r1,r2,..} ÷ {s1,s2,..}]S	select x.r3, ... from R x where set (x.r1, x.r2, ...) contains select y.s1, y.s2, ... from S y

Notes:

(1) R and S are relations over the domains {r1, r2, ...} and{s1, s2, ...}, respectively.

(2) The notations used for relational algebra follow that of Codd [Codd 71].

In so far as database accessing is concerned, the functions of the protocol are embodied in the network virtual data manager. We thus devoted a substantial effort analyzing of the nature of the task that an NVDM has to perform. Based upon this study, a general solution approach has been formulated (Kimbleton 79). Basically the strategy is to divide the tasks into two types: "user"-processing and "server"-processing. The user modules perform the functions of user catering: connecting users to DAP, accepting and interpreting raw inputs, and translating users" requests for database operations--opening databases for processing, retrieving selected portions from the databases, etc.--into commands acceptable to the individual remote DBMSs. The server modules perform the functions of interfacing to the individual DBMSs: initiating the execution of data retrieval requests and delivering the outputs from the DBMSs (in a meaning-preserving way) to the user station.

The decision to allocate functions in this particular fashion is based upon considerations of the DAP operating environment, namely, one involving a multiplicity of machines. The realization that these computers are not only heterogeneous but also widely dispersed geographically and administratively, convinces us that the implementation of our protocol must be highly target-adaptable. Hence, we decided to place all target -DBMS - independent tasks, that is, the bulk of the NVDM processing functions, into the user station. Among the target-unique functions, all query processings (The "dynamic semantic" phase of the translator (Kimbleton 80]) were also assigned to user-DAP. The amount of information shared between this module and other parts of the query translator precludes their physical separation. Attempts were made to minimize the target-dependency of their modules: their final output is not the actual target-DBMS query sequence, but a "tree" representation of it. That is, user-DAP generates "logical" (target-DBMS) queries--uncommitted as to the physical appearance of textual elements ("tokens"), but committed to the grammatical rules of sequencing and grouping them. (The readers are referred to (Kimblton 80) for detailed discussions of the query translation process.) The individual servers do the actual generation of the "physical" target-DMLs (from their tree representation), provide remote DBMS interfaces and are responsible for the needed data transfer operations. The small size and functional specificity of these servers makes it possible to build them with the high degree of extensibility and modifiability that we require. Thus, we let the user-DAP handle the "real" -NVDM processing--the translation of user requests into individual DBMS data manipulation language. The servers take essentially no part in the processing of database requests.

Besides potential improvements in target-adaptibility (Kimbleton 80), this "off-loading" of major NVDM functions onto the user station has other advantages as well. Specifically, it allows us to use the "family"-approach to the implementation of user-DAPs. That is, we can construct a single translator to perform those tasks common to all DBMSs. We can then build the target-specific portions of the translation process as post-processors to the common translator. This makes the implementation very extendible: the incorporation of new target DBMSs only requires the building of the particular target-oriented functionalities on top of the common framework, and need not be started from scratch.

To summarize, DAP is built from an interconnected set of stations, which are individual processes that coordinate their actions by means of message exchanges. The "user" processes, resident on the host to which users are directly attached, support terminal access to DAP services and perform all the essential NVDM processing functions on the virtual database. The server processes, resident on the DBMS hosts, provide local DBMS and Data Transfer Protocol (DTP) interfaces.

3.2 User-Server Interactions

Let us now look at a typical transaction between a user-DAP and the server-DAPs. At the request of the user (hereafter referred to as the "protocol invoker", to avoid confusion with other uses of the term), the user-protocol interpreter (user-PI) first decides to which host machine (s) the request is addressed. It then establishes control connections between the user station and the destination servers, to pass protocol commands and responses between the corresponding PIs. After this, the user-PI sets up data connections so that the servers can transmit the data retrieved by their local DBMSs back to the user-DAP. These individually retrieved portions of the databases are further aggregated and processed (by the user-DAP) before being delivered to the protocol invoker.

Figure 1 shows a situation somewhat more elaborate than that described above. It illustrates the case where a user wishes to access two different databases simultaneously. It might seem that cases like this are best handled by single multi-party interactions (Thomas 73). But this disregards the structure of one of the lower-layer services that DAP requires--the Data Transfer Protocol. As is evident from the discussions of (Wood 80) and(Wang 80), the distribution and assignment of the data translation/transformation functions of DTP does not allow an easy fit into the multi-party interaction models. Thus, DAP chooses the many-two-party interactions approach and Figure 1 shows that, for the case under consideration, two sets of control and data connections are needed for two separate (but concurrent) user-server interactions.

As described, DAP is essentially a request/response protocol: the sender waits until the receiver accepts and acknowledges a message before sending the next one. The only exception is that we have also made provisions for "interrupts" to be exchanged on a separate control connection. These interchanges are special messages that enable one station to alert another as to the occurrence of exceptional or unusual conditions. The flow and delivery of these interrupt signals are idependent of, and take precedence over that for normal messages (See Sec. 4.1 for further discussions)

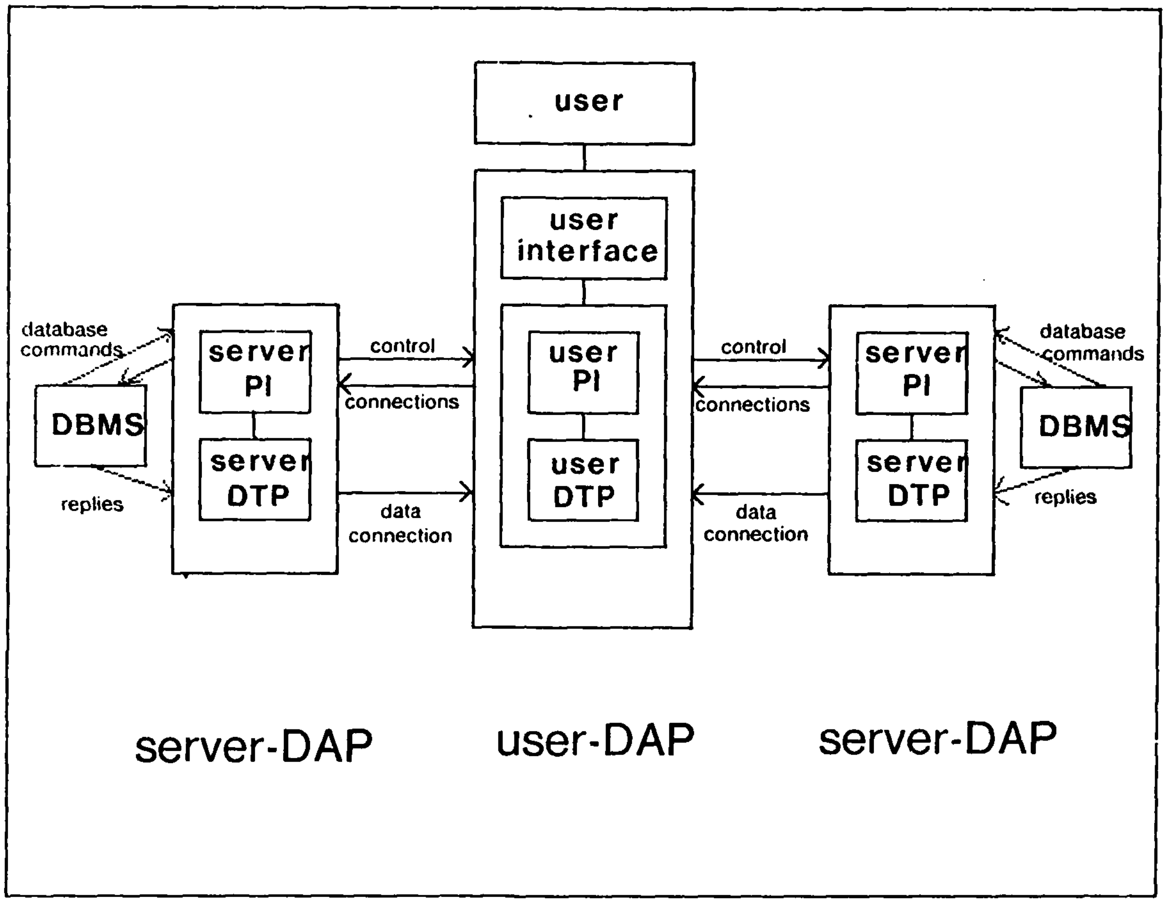


Figure 1. The DAP Protocol Model

Following the Arpanet file transfer protocol (Arpanet 78), we require that control connections be open while the message transfer is in progress in order to avoid losing or garbling messages in transit. When finished with DAP services, the user-PI sends a "terminate" message to each of the servers, which upon receipt of the request, take the actual action of closing the connections.

3.3 Relationship to Other Protocols

DAP is built upon certain lower level services: the TELNET protocol (Arpanet 78) which is used to gain access to remote hosts and to establish control communication between user and server processes, and the data transfer protocol (DTP) (Wood 80), (Kimbleton 80), (Wang 80) which is used to transmit data among heterogeneous systems while preserving its meaning. TELNET and DTP, then, determine the "transport" and the "coding" components (Sproull 78) of DAP(3). The relationship between DAP and other protocol layers is shown in Figure 2. In addition, it also requires a set of local DBMS facilities for its operation. This is the set provided by "typical" generalized database management systems (Codesyl 73), (Fry 76). A more detailed discussion of this last issue will be given in the next section.

4. A Description of the Protocol

The DAP protocol consists of two essential parts: messages controlling the actions of the user- and server-stations, and messages specifying the database management services to be performed at the remote hosts. The former are described in DAP as the "control commands", and the latter the "database commands".

4.1 DAP Control Commands

To support on-line conversational use, DAP provides commands for run-time execution control. Each command consists of two basic elements: the command name and a list of arguments on which the command operates. Each argument consists of an argument type identifier plus the parameter itself The commands are listed below in functional order. Each argument consists of an argument type identifier plus the parameter itself. All command descriptions are based on a common format: the command name, underlined, is given first with a very short description of its purpose. This is

(3) This is why only the language component of Sproul and Cohen's three components of a high-level protocol is discussed in this paper.

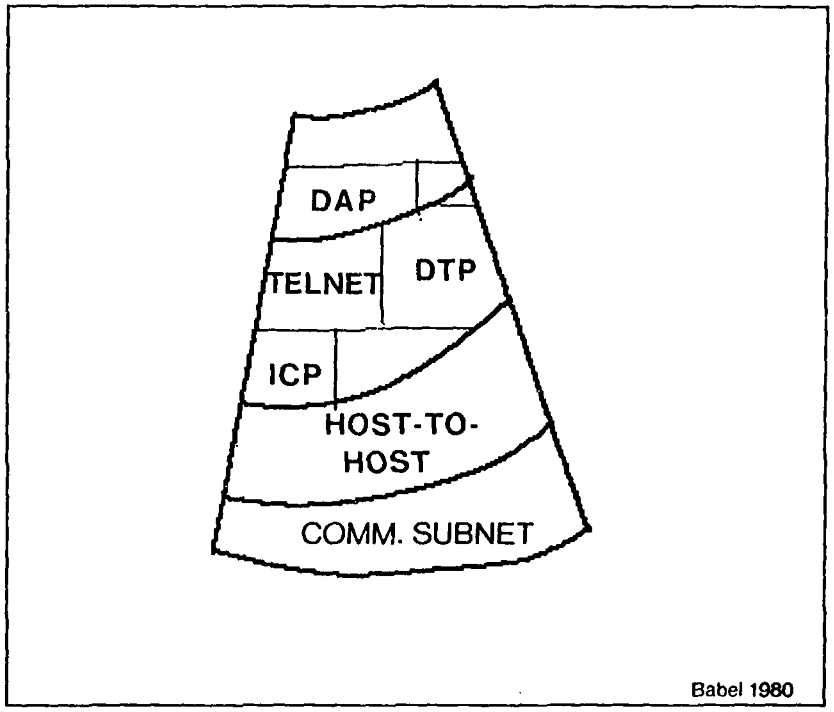


Figure 2. DAP as a Member of the Protocol Hierarchy

followed by the usage line, which summarizes how to use the command, where the following conventions apply:

1. Underlined words used to designate argument type are considered as literals.

2. Square brackets around an argument indicate that the argument is optional.

3. Argument names beginning with a hyphen, such as - name, are used to indicate the argument-type of the parameter that immediately follows.

(1) Access to the System

login	used to gain access to DAP
Usage:	login [-name Name] [-pass Password] "Name" is the name of user and "Password" is his password. If login is invoked without these arguments, it will ask for them.
logout	terminates a DAP session
Usage:	logout

(2) Conditional Execution Facilities

set-usage to define limits for command execution

Usage: set-usage -type Type -limit Limit where "Type" and "Limit" can be:

dollars	n
cpu	n
output	n

These specify that each command can consume, respectively, at most n dollars, n cou units or n lines of output

set-time to define time limits for command execution

Usage: set-time -before Time
or: set-time -after Time
These are used to execute commands in background, with the constraint that it should be before or after the (clock) time limit as given by "Time".

(3) Run-Time System Control

abort	interrupts the execution of the command
Usage:	abort The precise effect of this function, e.g., whether the processing is aborted or suspended, is left (for now) to the implementation (to accommodate the idiosyncracies of various operating systems).

4.2 DAP Database Commands

The DAP database commands may be categorized into those specifying database control functions (i.e. the NVDCL), those defining the Network Virtual Schemas (the NVDDL) and those for data manipulations (the NVUL) and for queries (the NVQL). Only queries and the most basic of the database control commands have been implemented. Therefore we shall limit the following discussions to this subset. The syntax is presented in Backus-Naur form (we have not attempted, nor is it relevant, to make the grannar unambiguous). The discussions on semantics and pragmatics serve to delimit the syntactically correct constructs which are meaningful. As an aid to the clarity of presentation, typographical conventions are adopted to distinguish the three kinds of constructs of the network virtual data languages: reserved words (e.g. select), non-terminal symbols (e.g. query) and identifiers (e.g. emp).

4.2.1 Database Control Language

The NVDCL provides the facilities for invoking and terminating remote DBMS services:

Syntax

```
<dcl - command> :: = <opendb-stmt> ! <closedb-stmt>

<opendb-stmt>   :: =  opendb <database-name>  -mode <mode>

<closedb-stmt>  :: =  closeb <database - name>

<database-name> :: = <identifier>  -dbms <dbms-name>

<dbms_name>     :: = <identifier> @ <host_name>

<host_name>     :: =  <identifier>

<mode> :: = c ! er ! u ! eu
```

Semantics

The opendb command starts a session with a remote DBMS: it establishes connections to the remote host and opens the appropriate database. Thereafter the system will accept other database commands and process them in the designated usage mode. There are four such possible modes:

r retrieval with concurrent access (both for retrieval and update) to the database allowed.

u update with concurrent access (both for retrieval and update) to the database allowed.

er exclusive retrieval prohibiting concurrent access for update purposes

eu exclusive update prohibiting concurrent access (for either retrieval or update).

Closedb "logs off" the user from the specified database (makes it unavailable for processing) and closes the connections to the remote DBMS host.

Pragmatics

It might seem that automatic starting and stopping of remote DBMS sessions is a useful user service. Nevertheless, considerations of implementation issues persuaded us against it. The cost (in terms of amount of processing and the "slot" it occupies as one of the concurrent users of a DBMS of keeping a database open demands a highly optimized, dynamic mechanism. And we feel that such automatic open/close mechanisms are still beyond the current state-of-the-art (at least, our own) capability.

By concatenating the location (host name) of a database to the DBMS name, we expanded the naming hierarchy with a top most level. This embeds (a very elementary form of) the distribution management problem into the naming scheme. This is the same approach as that used in RSEXEC (Thomas 73) Alternative ways for specifying target databases, for example, "implicit" specifications, have been proposed (Kimbleton 79) and are being studied. This is the case where DAP maintains sufficient information to identify the relevant databases, so the user can issue commands without any target database specification. More powerful ways of devising names (and hence, for handling distribution issues) (Saltzer 78), (Watson 80) are also being investigated.

As remarked before, DAP treats such functions as addressing and connection management as an integral part of database control-addressing is performed when database names are interpreted and connections are established or broken at the time databases are opened or closed. This came about from the expectation that typical user-server communication would be based on a single DBMS session (which of course, usually consists of a sequence of several messages). The performance implications of this decision remains to be explored.

4.2.2 Database Query Language

The NVQL provides the following elementary operations on the basic NVS construct--i.e. tables:

(1) Projection

Syntax

```
<query>  ::= <sel_clause>  from  <from_list>

<sel_clause> ::=  select <sel_list>

<sel_list> ::= <sel_expr>  ! <sel_list> ,<sel_expr>

<sel_expr> ::= <expr> !<var_name> .*

<from_list>  ::= <table_name><var_ name>
              ! <from_list> , <table_name><var_name>

<expr>  ::= <expr>  <add_op> <expr>
            ! <expr> <mult_op><expr>
            ! <add_op><expr>
            ! <primary>

<primary>  ::= <field_spec>  ! <constant>(<expr> )

<field_spec>  ::= <var_name> . <field_name>

<add_op>  ::=  + ! -

<mult_op> ::=  * ! /

<constant> ::= <quoted_string> ! <number> !  null

<table_name> ::= <database_name> . <identifier>
             ! <identifier>
<var_name>  ::= < identifier>

<field_name> ::= <identifier>
```

Semantics

This operation retrieves the set of values in the columns selected in sel_list (or arithmetic expressions constructed from them) from the relations named in from_list . The *-notation for field names allows us to retrieve the entire relation without explicity specifying all column names.

Like SQL, duplicates are not removed from the returned set of values. But the reason for this decision is not to economize on processing cost (as is the case for SQL (Chamberlin 761). Rather it is our desire to attain generality for the protocol. This point needs some further explanation. We believe that programmers, especially writers of interactive programs, usually need to view what constitutes "duplication" differently from other users and from the DBMS(4). This observation arose from our experience with the implementation of a prototype server-DAP for the Honeywell relational database system MRDS (Honeywell 78), which automatically removes all duplicates (tuples that are equal to each other, data-item by data-item) from the retrieved result. There were numerous occasions when we had to detour around this feature since for our application the "duplicates" are in fact different tupes which happen to"look alike". However, the many implementation issues implied by this decision still remain to be studied.

The SQL feature of allowing the existence of unknown (null)values in the data base is included in NVQL: this is highly desirable since it allows parts of a distributed database to differ in their information contents. It provides the needed "escape" mechanism for us to handle situations where differerent portions of the database are inconsistent or incomplete with respect to each other.

Example

List the employee number, name and salary of all employees.

select x.empno, x.name, x.sel from emp x

Pragmatics

Although retrieval of computed values (arithmetic expressions) is not necessary in a query language, such a capability is highly desirable, especially for NVQL, since it enhances the power of query composition (see (5) below) in a very substantive way.

(2) SET ABSTRACTION

Syntax

```
<query> ::= <sel_clause> from <from_list>
            where <boolean>

<boolean> ::= <boolean> and <boolean>
            ! <boolean> or <boolean>
            !  not <boolean>
            !  <pred>

<pred> ::= <expr><comp_op><expr>
            !(<boolean>)

<comp_op>::= > ! >= !  = ! ^=  ! < ! <=
```

Semantics

(4) Similar situations arise with programming language-query language interfaces, which led to the introduction of "cursors" in SQL.

From the relations named in from_list , this query retrives the set of values of the columns selected in sel_list (or arithmetic and aggregate values computed from them) where the associated tuples satisfy the boolean condition given in the "where" clause.

Example

Find the names of employees in Dept. 50.

```
select x.name from emp x
where x.dno = 50
```

For each employee whose salary exceeds his manager's salary, list the employee's name and his manager's name.

```
select x.name, y. name from emp x, emp y
where x.mgr = y.empno and x.sal > y. sal
```

Pragmatics

For reasons given later in (6), the set operations union, intersection and difference are needed in NVQL, but we still allow conjunctions and disjunctions in the logical expression used for set abstraction, because replacement of these with set operations are both inefficient (if proper care is not taken by the query optimizer) and unnatural.

(3) AGGREGATE FUNCTIONS

Syntax

```
<primary> ::= <ag_fn> (<field_spec> )
<ag_fn> ::= avg ! count ! sum ! max ! min
```

Semantics

An aggregate function is a mapping that assigns a (scaler) value to a set of tuples. In NVQL, we choose to build in the (in our opinion) most commonly used aggregate functions, that is, the average, count, sum maximum and minimum of a set of numbers (i.e. tables with a single numerical-valued column).

Example

Find the average salary of clerks.

```
select avg (x.sal)  from emp x
where x.job =  "clerk"
```

Pragmatics

They are included in NVQL for the same reason as that for arithmetic expressions (see (1) above).

(4) PARTITION

Syntax

```
<query>  ::= <sel_clause>  from  <from_list>
             [where <boolean>]
             [<group_clause> ]

<group_clause> ::=   group by <field_spec_list>
                     [having <group_boolean>]

<group_boolean> ::=  <group_boolean> and  <group_boolean >
               ! <group_boolean>  or  <group_boolean>
               !  not <group_boolean>
               ! <group_pred>

<group_pred> ::= <ag_fn> (<field_spec> ) <comp_op><expr>
    !  set  (<field_spec_list> ) <set_comp> (<field_spec_list> )
    !  ( <group_boolean > )

<field_spec_list> ::= <field_spec>
               ! <field_spec_list> , <field_spec>
```

Semantics

The relation is broken up (partitioned) into non-overlapping subsets (groups of tuples) such that each tuple in a subset has identical values in the columns specified in the "group by" clause. A new relation (the partitioned relation) is then constructed to describe the properties of these groups: for each column of the original relation, this consists of the set of elements of the column in all the tuples of the group and their aggregate values (whenever applicable).

Example

List all the depart ments and the average salary of each.

```
select x.dno,  avg (x.sal)  from emp x
group by  x.dno
```

List those departments in which the average employee salary is less than 1000.

```
select x. dno from emp x
group by x.dno
having avg (x. sal)< 1000
```

Pragmatics

Conceptually a partitioned relation is a relation over groups of tuples and therefore only contains attributes that are relevant to the groups and not the individual tuples. This semantic interpretation is reflected in the syntax of the group qualifier (the "having" clause). The order in which the various operations in a query are executed is the same as that for SQL: First the "where" clause is applied to qualify tuples: then the groups are formed; and the "having" clause is applied to qualify groups and finally the selection is carried out to retrieve the selected data items.

(5) COMPOSITION

Syntax

```
<pred> ::= <expr> <set_comp> <table_spec>
   !  <field_spec_list> > <set_comp>  <table_spec>

<group_pred>  ::=  set  (<field_spec_list> )
                          <set_comp> <table_spec>

<table_spec>  ::= <query>
               !  ( < query > )
               !  < literal>

<set_comp> ::=  contains
             !  does not contain
             !  [is]  in
             !  [is]  not  in
             !  =
             !  ^=

<literal>  ::=  ( < lit_tupe_list > )
              !  <lit_tuple>
              !  (<constant_list > )
              !  <constant >

<lit_tuple_list>  ::= <lit_tuple >
               ! <lit_tuple_list > , <lit_tuple>

<lit_tuple> ::=   < constant_list >

<constant_list> ::= <constant>
               ! < constant_list > , <constant>
```

Semantics

This SQL facility of allowing the user to compose a query by applying an operation to the result of another is adopted by NVQL. This technique is useful since it makes the description of three- four-, and more-place associations by means of binary relations (tables) very easy.

Like SQL, all set comparison operations eliminate duplicates from both of their operands before performing their operations.

Example

Find the names of employees who work for departments in Evanston.

```
select x.name  from emp x
where x. dno in
              select y. dno  from dept y
              where y.loc = "Evanston"
```

List the suppliers that supply all the parts used by Dept. 50.

```
select x.supplier from supply x
group by x. supplier
having set (x. part) contains
              select  y. part  from  usage y
              where  y.dno =  50
```

(6) TRADITIONAL SET OPERATIONS--union, intersect, minus (set difference)

Syntax

```
<query> ::= <query_expr>
          ! <query_expr><set_op><query>
          !  (<query_expr> )

<set_op>  ::=  union  ! intersect  ! minus
```

Semantics

These operations have their conventional set-theoretical means.

Example

List the departments that have no employees.

```
select x.dno  from dept  x
minus select  x.dno  from emp  x
```

Pragmatics

Intersect and minus are not necessary since we can re-express the intersection of two relations R, S (with column headings col1, col2,..., colN), as

```
select r. * from R r
where r.coll, r. col2..., r. colN in
select s. * from S s
```

union can not be obtained from set abstraction since it produces relations whose domains are supersets of the domains for the operand relations. (Set abstraction always results in smaller relations than what we started out since since we do not have the "is a member of" type of predicate that Codd's ALPHA (Codd 71) has.) But all three operations are included for uniformity, and also in some cases, we believe intersect and minus are easier to use than their set-abstraction equivalents.

Like the set comparison operations of (5), union, intersect and minus automatically eliminate duplicates from their operands before performing their operations.

5. SCOPE OF THE WORK

As the above discussion shows, a database access protocol is complex in the many issues it must address. Figure 3 attempts to show the different dimensions along which data base systems may vary, and the types of management tasks associated with them.

A database system is used and manipulated in a number of different ways. In the diagram, this is compressed in one dimension labelled as "DBMS category". Databases systems may be characterized

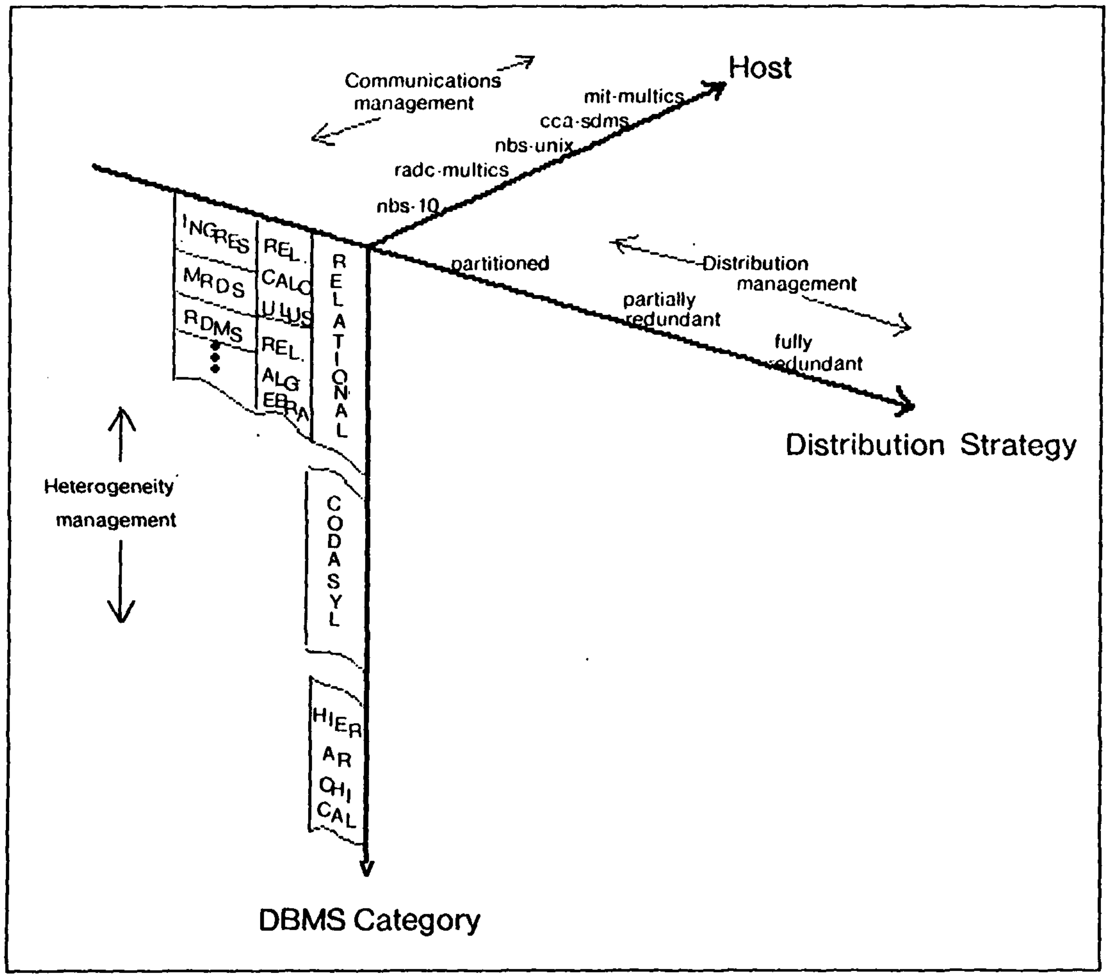


Figure 3. Dimensions of Variation of Database Systems

according to the various data models they adopt. (Only the three best known approaches are illustrated in the diagram--the relational, the Codasyl DBTG and the hierarchical data models.) Each system also differs from all others--in the ways data structures are manipulated, in the syntax of their data languages, in their choice of keywords for the various commands, and in their operational characteristics (Date 77).

Holding the DBMS-category constant, we can take several different approaches to the distribution of data. Each database site could contain a complete copy of the entire database (fully redundant); or each site could contain a unique subset (partitioned). Yet another choice would be to allow some overlap of data among the different sites (partially redundant) (Rothnie 77). This is illustrated in the diagram as the "distribution-strategy" dimension.

The databases are also located physically on different machines, this is given as the "host" dimension in Figure 3. (Since our current implementation uses the ARPANET (Arpanet 78) for communication support, we have chosen to set Figure 3 within this context, too. Therefore the labels used for the "host" dimension are the ARPA host-names--this is how they are recognized by the underlying IMP communications subnet (Heart 701).

Our design effort so far has been concerned primarily with handling the heterogeneity of the various DBMSs, only a little with communications management, and not at all with distribution strategies. The reason for this particular choice of focus is two fold. In the first place, we are aiming at an external environment of multiple, existing databases managed by a diversity of DBMSs, where heterogeneity stands out as the most difficult and the most important problem. Secondly, several major R&D efforts have already been directed at the data distribution issue (Alsberg 76), (Bernstein 80l, Ellis 77), (Rosenkrantz 79), (Stonebraker 79), (Thomas 79). An excellent survey of these developments is given in (Rothnie 77).

6. CONCLUDING REMARKS

In this paper we have endeavored to describe a protocol for facilitating the sharing of databases for remote users of a computer network. Considering the increasing interest and development in the general subject of distributed databases (Rothnie 77), (Gray 79), we feel that the problems and issues involved in our design process should be presented, even though the protocol is still at its early formulation stage, and many considerations are incomplete. To wit, both the service and interface specifications have been left untouched. Likewise, the distribution management aspect of the protocol has been barely outlined, and only the most elementary of such mechanisms (see Sec. 4.2.1) have been incorporated.

The main source of experience with the protocol so far is in the aforementioned prototype implementation, where we have constructed the user- and server-PIs for the Honeywell Multics Relational Data Store

(Honeywell 78), a relational-calculus system, and the Honeywell Integrated Data Store (Honeywell 71), a Codesyl-DBTG type system. (We emphasize that, as has been pointed out in Sec. 2.1, our current user-DAP is limited in that it only performs the NVDM to target-DBMS translation. The user-query to NVDM translation capability has yet to be constructed. Thus, for now, users are forced to interact with the system in NVQL.) A preliminary discussion of this work has already been published (Kimbleton 79) and a more complete treatment is forthcoming (Kimbleton 80). The protocol will undoubtedly undergo many re-designs in the months ahead. We hope the presentation of this paper will bring us the criticisms and reactions that we need to improve the design.

REFERENCES

(Alsberg 76) Alsberg, P.A. and Day, J.D. A Principle for Resilient Sharing of Distributed Resources, Proc. 2nd Intern, Conf. Software Eng. 1976, pp. 562-570.

(Arpanet 78) Arpanetet Protocol Handbook, NIC 7104, Network Information Center, SRI International, Menlo Park, Calif., 1978.

(Astrehan 76) Astrehan, M.M., Slasgen, M. W., Chamberlin, D. D., Eswaren, K. P., Gray, J. N., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. System R: Relational Approaches to Database Management, ACM Trans. Database Sys., Vol. 1 (1976), 97-137.

(Bernstein 80) Bernstein, P. A., Shipman, D. W. and Rothnie, J. B. Concurrency Control in a System for Distributed Databases (SDD-1), ACM Trans. Database Sys., Vol. 5 (1980), 18 - 51.

(Chamberlin 76) Chamberlin, P.D., Astrehan, M. M., Eswaran, K. P., Griffiths, P. P., Lorie, R. A., Mehl, J. W., Reisner, P. and Wade S. W. SEQUEL2: A Unified Approach to Data Definition, ManipuL2ation, and Control, IBM J. Res. & Develop., Vol. 20 (1976), 560-575.

(Chen 76) Chen P. P.-S. The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. Database Sys., Vol. 1 (1976), 9-36.

(Codasyl 71) Codasyl) Systems Committee, Introduction to "Feature Analysis of Generalized Data Base Management Systems". Comm. ACM, Vol. 14 (1971), 308-318.

(Codasyl 33) Codasyl Data Description Language Committee, DDL Journal of Development (1973).

(Codd 71) Codd, E. F. A Relational Model of Data for Large Shared Data Banks, Comm. ACM, Vol. 13 (1970), 377-387.

(Codd 71) Codd, E. F. Relational Completeness of Data Base Languages. In "Data Base Systems", R. Rustin, Eds., Prentice-Hall, Englewood Cliffs, New Jersey, 1971, pp. 65 - 96.

(Crocker 72) Crocker, S., Heafner, J. F., Metcalfe, R. M. and Postel, J. B. Function-orient ed protocols for the ARPA Computer Network. Procl. AFIPS SJCC, Vol. 40 (1972), 271-198.

(Date 77) Date, C. J. "An Introduction to Database Systems", 2nd Ed., Addison-Wesley, Reading, Mass., 1977.

(Deneffe 74) Deheneffe, C., Hennebert, H. and Paulus, W. Relational Model) for Data Base. Proc. IFIP Congress 1974, North-Holland, Amsterdam, pp. 1022-1025.

(Dijkstra 76) Dijkstra, E. W. A. Djacipline of Programming. Prentice-Hall, Englewood-Cliffs, N.J., 1976, Chap. 27.

(Ellis 77) Ellis, C. A. A Robust Algorithm for Updating Duplicate Databases. Proc. Second Berkeley Workshop on Dist. Data Mgmt. and Comp. Net., 1977, pp. -

(Fong 80) Fong, L. N. and Kimbleton, S. R. A Semantic Integrity System for a Network Data Manager, to be published in Proc. AFIPS NCC, 1980.

(Fry 76) Fry, J. P. and Sibley, E. H. The Evolution of Data-Base Management Systems, ACM Computing Surveys, Vol. 8 (1976), 7-42.

(Gardarin 77) Gardarin, G. and Le Bihan, J. An Approach towards a Virtual Data Base Protocol for Computer Networks. Proc. AICA 77, Italy, Calcolo Automatico, Milano, Italy.

(Gray 77) Gray, J. N. A Discussion on Distributed Systems. IBM Res. Rep. IBM Res. Lab., San Jose, Calif., 1979.

(Hainaut 74) Hainaut, J. L. and Lecharlier, B. An Extensible Semantic Model of Data Base and its Data Language, Proc. IFIP Congress 1974, North-Holland, Amsterdam, pp. 1026 - 1030.

(Heart 70) Heart, F. E., Kahn, R. E., Ornstein, S. M., Crowther, W. R. and Walden, D. C. The interface message processor for the ARPA computer network. Proc. AFIPS SJCC, Vol. 36 (1970), 551-567.

(Hevner 79) Henver, A. R. and Yao, S. B. Query Processing in a Distributed Database. IEEE Trans. on Software Engineering, Vol. SE-5, (1979), 177 - 187.

(Honeywell 71) Honeywell Information Systems "Integrated Data Store, Order No. Br69, Re. 1, Dec. 1971.

(Honeywell 78) Honeywell Information Systems "Multics Relational Data Store (MRDS) Reference Manual", Order No. AW53, Rev. 2, Oct. 1978.

(Kimbleton 79) Kimbleton, S. R., Wang, P. S.-C. and Fong, E. XNDM: An Experimental Network Data Manager. Proc. Fourth Berkeley Conf. on Dist. Data Mgmt. and Comp. Net., Aug. 1979, pp. 3-17.

(Kimbleton 80) Kimbleton, S. R. and Wang, P. S.-C Data Access Protocols: Objectives, Principles and An Implementation Approach, to be published in "Distributed Systems - Architecture and Implementation", Springer-Verlag, New York, 1980.

(Parnas 72) Parnas, D. L. On the Criteria to Be Used in Decomposing Systems into Modules. Comm. ACM, Vol. 15 (1972), 1053-1058.

(Rosenkrantz 78) Rosenkrantz, D. J., Stearns, R. E. and Lewis, P. M. System Level Concurrency Control for Distributed Database Systems. ACM Trans. Database Sys. Vol. 3 (1979), 178 - 198.

(Rothnie 77) Rothnie, J. B. and Goodman N. A Survey of Research and Development in Distributed Database Management. Proc. 3rd Intern. Conf. Very Large Data bases, Tokyo, Japan, 1977, pp. 48 - 62.

(Saltzger 78) Saltzger, J. Naming and Binding of Objects. In "Operating Systems - an Advanced Course", G. Goos and J. Hartmanis, Eds., Springer-Verlag, New York, 1978, pp. 100 - 208.

(Selinger 79) Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A. and Price, T. G. Access Path Selection in a Relational Database Management System. Proc. ACM SIGMOD, May 1979, pp. 23 - 34.

(Sproull 78) Sproull, R. F. and Cohen, D. High-Level Protocols, Proc. IEEE, Vol. 66 (1978), 1371 - 1386.

(Stonebraker 79) Stonebraker, M. Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES. IEEE Trans. Software Eng., Vol. SE-5 (1979), 188 - 194.

(Sunshine 79) Sunshine, C. Formal Techniques for Protocol Specification and Verification, Computer, Vol. 12 (1979), 20 - 27.

(Thomas 73) Thomas, R. H. A Resource Sharing Executive for the ARPANET, Proc. AFIPS NCC, Vol. 42 (1973), 155 - 163.

(Thomas 79) Thomas, R. H. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, ACM Trans. Database Sys., Vol. 4 (1979), 180 - 209.

(Wang 80) Wang, P. S.-C. and Kimbleton, S. R. A Data Transfer Protocol for Remote Database Access. to be published in Proc. NBS-IEEE Trends and Applications Symp., 1980.

(Watson 80) Watson, R. Naming in Distributed Systems, to be published in "Distributed Systems - Architecture and Implementation", Springer-Verlag, New York, 1980.

(Wong 77) Wong, E. Retrieving Dispersed Data from SDD-1: A System for Distributed Databases, Proc. Second Berkeley Workshop on Dist. Data Mgmt. and Comp. Net., Aug. 1977, pp. 217 - .

(Wood 80) Wood, H. M. and Kimbleton, S. R. Remote Record Access: Requirements, Implementation and Analysis, NBS SP - (in preparation), National Bureau of Standards, Washington, DC. 1980.

AD-A102 776   NATIONAL BUREAU OF STANDARDS WASHINGTON D C   INST FOR--ETC   F/6 9/2
XNDM: AN EXPERIMENTAL NETWORK DATA MANAGER.(U)
JUN 81   S R KIMBLETON, P S WANG, E N FONG   F30602-77-C-0066
UNCLASSIFIED   RADC-TR-81-123   NL

5 OF 5
AD
A102776

END
DATE
FILMED
9-81
DTIC

APPENDIX C

THE EFFECTS OF COMMUNICATIONS SUBNET CHOICE UPON HOST-HOST PROTOCOL PERFORMANCE

Leslie Jill Miller
State University of New York at Brockport
National Bureau of Standards

Stephen R. Kimbleton
National Bureau of Standards

May 1980

ABSTRACT

Connection oriented host-host protocols can be built above networks which provide a simple datagram service, or above connection-oriented X.25-like networks. A model host-host protocol is defined, and those performance factors which are affected by the subnet choice are investigated. In particular, packet overhead for connection maintenance and host buffering requirements are shown to be essentially identical for the two types of subnetwork, while throughput loss due to packet headers is significantly higher for datagram networks. TCP is compared to the model host-host protocol, and its throughput degradation due to header overhead is quantified.

1. INTRODUCTION

1.1 Host-host Protocol and Network Services

For most distributed applications there is a need for reliable, sequenced, and flow-controlled transfer of data. This service is generally provided by a transport protocol [1] embodied in a a host-host protocol, where the usual implementation involves the creation of a virtual circuit between the two host level application processes. Within this connection the host-host protocol monitors and corrects the flow of data across the underlying communication subnetwork.

The subnetwork itself must provide the necessary addressing and routing capabilities to get packets of data from one network location or network connection endpoint to another. The transfer is not necessarily error-free, sequenced or flow-controlled, but the network may attempt to provide some of these services.

The Department of Defense has developed a host-host protocol called TCP (Transmission Control Protocol) [2]. It assumes that the underlying network provides very few services — mainly routing, addressing, and fragmentation, if necessary — of independent packets called datagrams. The internetwork protocol which provides this low level datagram service is called the Internet Protocol, IP [3].

In contrast to IP, the international standards organization CCITT has concurrently been developing a standard interface, called CCITT Recommendation X.25 [4], for public packet switched networks, which demands much more from the underlying network. Without requiring network packet retransmission, it tries to provide many of the host-host services. Packets are flow-controlled (on each logical channel) at the interface to the network, packet sequencing is maintained, and errors result in end-to-end resets of the logical channel. The interface specification is such that a very efficient fixed-routing network implementation is possible, and most public networks use such an implementation.

1.2 Performance Evaluation

In this paper we define a model of a general host-host protocol, and examine its performance as affected by the underlying communications subnet. Since we are not studying host-host protocols in any general sense, we consider a fairly minimal set of host-host protocol features. We specifically examine the packet overhead required to establish network level connections, the effects of different packet header lengths in diminishing useful throughput, and the required number and occupancy of buffers. The specific subnets which which we compare are

(1) a datagram network similar to that defined by the IP protocol, and (2) a connection oriented X.25 network having a fixed routing implementation. The internal characteristics of these two networks are described in Section 2.4.3. We also compare TCP to our model host-host protocol, and we consider those performance measures which are affected by its being based on a datagram network.

We conclude that somewhat better throughput for a host-host protocol can be obtained if the underlying network is assumed to provide (unreliable) connection oriented service. In this paper we do not handle the issues of internetworking which further support this view but they will be considered in a later paper.

2. PROTOCOL MODEL

2.1 Protocol, Service and Interface Specifications

In a recent paper on protocol specification and verification [5], Sunshine characterizes three necessary specifications for each level in a protocol heirarchy. One must first specify the service features of each level, thus defining the externally perceptible features of that level, as we have superficially done for the network and host-host levels in our introduction. One must also define a specific interface for each level, as is done, for example, by the precise specification of the X.25 interface with its description of each field for every packet type. Finally, one must describe the interactions between the distributed components of the particular layer which provides a given service, as is done for example in describing the header formats and processing algorithms in TCP. These three types of specification are referred to as being the service, interface and protocol specifications of a given layer in a distributed system.

In order to compare the performance effects of different subnetwork choices, we describe those features of a basic host-host protocol which depend upon the choice of subnet, and detail the ways by which the two different subnets provide their characteristic service. We start by describing the natural data elements for each protocol level, and then briefly consider the differences between host-host and network level connections. Having completed the preliminaries, we specifically consider the protocol features at each level which contribute to reliable, sequenced, flow-controlled data transfer.

2.2 Protocol Information Units

The basic means of ensuring reliability in a distributed system involves an acknowledgement - retransmission scheme whereby an information unit is sent to some destination, a backup copy is retained until an acknowledgement is received, and if the acknowledgement is not received within some fixed time interval then the packet is retransmitted. This positive acknowledgement scheme is typically used at each of several layers in a protocol heirarchy, with successively larger information units being acknowledged at each higher level.

We will refer to the basic data unit within a network as being either a packet or a datagram. Its size is usually constrained to be less than some maximum which is chosen so as to simultaneously:

1. minimize the total fraction of transmitted bits which consist of header information. (This favors big packets.)

2. minimize the effects of having to retransmit corrupted packets. (This favors small packets.)

3. minimize internal fragmentation of buffer space (This favors small packets.)

4. minimize packet processing (This favors big packets.)

Typical implementations choose maximum packet sizes of 500 - 4000 bits. Each packet is then wrapped in a separate link-level frame for transmission. The link level protocol relies on frame CRCs (cyclical redundancy checks) and frame retransmission to reliably get the packet across the link to the next node.

At the top of the protocol heirarchy, an application process typically wants to submit arbitrarily large information units, which ISO (the International Organization for Standardization) refers to as quarantine units [1], to the host-host protocol for transmission to a remote process. The application process expects that these quarantine units will be reliably delivered as long as there is some path through the underlying network, assuming that neither host crashes with loss of memory.

It is the responsibility of the host-host protocol to see that the quarantine units are transmitted across the network as a stream of packets. While frame transmission across individual links can be made as accurate as desired, packets may be discarded or corrupted at network nodes, and network components may crash. The host-host protocol manages the necessary end-to-end error detection and correction in units, called segments, which are typically larger than a single packet. Segments are usually larger than packets since every segment must carry header information which is used by the host-host protocol, and the effects of this overhead should be minimized. Since error rates are low after link level retransmission, segments are retransmitted infrequently and the host-host protocol can afford to retransmit whole segments, some of whose packets may have already been received correctly.

We thus see that the network, the host-host protocol, and the application process each have a natural data unit, one whose maximum length is optimal with respect to the conflicting desires to minimize the effects of header overhead, to optimize buffer management and to minimize the amount of unnecessarily retransmitted information.

2.3 Network versus Host-host Protocol Virtual Circuits

Virtual circuits established by the host-host protocol should be about as reliable as the host machines at their endpoints. In particular, errors in the network should be correctible as long as the source and destination are connected. This may involve reestablishing network level virtual connections in the case of internal network failure.

Because logically distinct data from many virtual circuits will be sharing the same physical transmission paths, each connection must be uniquely identified within the connected networks, and each segment should carry this identification explicitly or implicitly.

Data within a host-host virtual circuit must be sufficiently described so that the host-host protocol can determine whether or not all of a quarantine unit has arrived at the destination, and whether it is in exactly the same order as it was when submitted by the sending process. Depending upon the subnet, it is possible for packets to arrive out of order as a result of both packet retransmission and differences in delay within the network.

The network virtual circuit differs from the above characterization in several respects. It is not expected that the network can always repair broken connections, but the network should always inform the host-host protocol of a break. The connection number only has to be unique within a single network or interface, and packets only need sufficient description to determine whether they are in correct order. It is also not expected that a network will provide packet retransmission, and thus packet resequencing will be either fairly fast or impossible.

2.4 Protocol Feature Analysis

The services provided by the host-host protocol will usually consist of a set of operating system-like calls to open and close connections, and to send or receive buffers or files of data. Buffer management will depend upon the operating system but generally the transmitting process of the host-host protocol will retain control of a data buffer until the destination process acknowledges the data's arrival, while the receiving host-host protocol process will only pass data to the application process after a full quarantine unit has been correctly received.

Since our main concern is with the relationship between subnet service and host-host protocol performance, and not with a study of host-host protocols in general, we will only consider a minimal set of essential host-host protocol features. We delay until Section 4 our consideration of the specific host-host protocol TCP.

2.4.1 Host-host Protocol Features

There must be some provision for the following fields in the protocol header. Approximate field lengths are given for later comparison purposes.

1. Connection Number: The concatenation of the source and destination process addresses gives a unique connection number which is easily determined in a distributed environment. Allowing for a 32 bit internetwork host address and a 16 bit local process number, we require 96 bits for the connection number.

2. Sequence Number, Acknowledgement Number and Window Size: These numbers should ordinarily be expressed in terms of segments. The number space must be sufficiently large that a long-lived segment from a previous incarnation of the connection cannot be mistakenly considered as a valid segment within the current incarnation of the connection. The window size is an end-to-end indication of the maximum allowable number of outstanding segments. We will assume that 16 + 16 + 8 bits are sufficient for these three fields.

3. Segment Length: To check that all data within a segment has arrived requires a length field, which we will express in octets. A 16 bit field is sufficient.

4. Checksum: Another end-to-end check for data validity is to calculate either a checksum for the entire segment, including header, or a CRC. We will assume a 16 bit checksum field. We specifically include the connection number within the data being checked, so as to verify that the segment has not been misdelivered.

5. Control Bits and Format Designator: We will allow 8 bits.

The above list of features characterizes a very basic host-host protocol which uses a header of 22 octets or 176 bits.

2.4.2 Network Interface

The network user must provide the following information on each packet or datagram submitted to the network.

1. Address Information: We will assume that every packet or datagram must carry a designation of both the sending and the receiving hosts. In a connection oriented network, similar to X.25 but not using its X.121 international numbering plan, only the call establishment packets need contain the 64 bits of the two host addresses. The data packets in an X.25 network carry just a 12 bit connection number for address designation across the network interface. In a datagram network every datagram must carry the two host addresses.

2. Type of Service: We will allow 8 bits for designating the desired type of service. Again, this field is required on every datagram but only required in the call establishment packet of a connection oriented network.

3. Flow Control: To allow for efficient flow-control across the interface we allow for 8 bits to encode sequence and acknowledgement numbers.

4. Packet Identification: There must be some way of identifying packets as they cross the network interface, in order to later sequence them and determine whether a packet is missing. In an IP datagram this consists of a unique number for each segment plus a fragment offset, for a total of 32 bits. In an X.25 network with fixed routing the sequence number (modulo 8) together with the connection number serve this purpose.

5. Control Bits and Format Designator: We again allow 8 bits.

2.4.3 Network Protocol

Within the network itself each packet can be handled as an independently routed datagram, and segments can be reassembled by the host-host protocol at the destination. The other alternative we consider is that the packets are sent along virtual circuits which are implemented via fixed routes.

Even networks supporting an X.25 interface can be implemented internally using datagrams, with the resequencing which does not require packet retransmission being done within the network at the destination node. In this case, each explicitly addressed packet or datagram within the network must carry all the information which is presented at the network interface, except possibly the interface flow-control information. Because of the possibility of loops in a dynamic routing algorithm, however, the datagram must instead carry a Time to Live field which is periodically decremented. When it reaches zero the datagram is discarded. Thus the datagram in general requires a header of 15 octets or 120 bits.

Alternatively, an X.25 network can be implemented so as to use fixed routing. By defining 2**12 logical channels on each physical link, it is possible to fix a logical path through the network. During call establishment the best route through the network is dynamically determined, and a logical fixed connection is established. For a packet later coming into a node on link 10 with logical channel number 245, the node can then determine that the packet should leave via channel 178 on link 3, and after altering the packet's logical channel number (from 245 to 178), the packet is forwarded. By using this fixed routing one cuts the length of the address field from 64 bits to 12.

Furthermore, it is impossible under this fixed routing scheme for packets to get out of order, though packets may get discarded. The small sequence space used for flow-control over the interface will thus in most cases be sufficient to detect missing packets. There is no need for resequencing buffers within the network with this implementation scheme, and the Time to Live field is unnecessary.

There are, however, problems with a fixed routing implementation of a connection oriented network. Internal errors may result in inefficiently rerouted connections, which may even cause packets to partly retrace their path. Also, with fixed routing the network loses the ability to route individual packets around dynamically changing points of local congestion, though some nets may reroute connections when congestion on the original route gets bad. Thus total packet throughput across the net may be decreased. The effect of these drawbacks is small, however, and all current public packet switched networks do employ some form of fixed routing when they provide the X.25 interface.

The data packet header used internally in a fixed routing X.25 network need only contain the packet format information, the logical channel number, and the interface flow-control sequence and acknowledgement numbers, for a total of 3 octets or 24 bits.

3. HOST-HOST PROTOCOL PERFORMANCE

In this section we make a performance comparison between a host-host protocol based on a datagram subnet, and a host-host protocol based on a network having both an X.25 interface and the fixed routing implementation which the interface facilitates.

We specifically consider the overhead involved in setting up network connections, the reduction in throughput attributable to packet and segment headers under the two schemes, and the differences in buffering requirements.

3.1 Connection Overhead

The establishment of a host-host virtual circuit involves a very careful synchronization of segment numbers, to prevent "old" packets or segments from previous incarnations of a connection from being considered as valid. A three way handshake is the TCP implementation technique. The establishment of an X.25 connection is much less careful, and each time a connection is reestablished or reset, packets are numbered starting at 0 again. This network connection establishment involves just a two way handshake.

Since the call establishment packets in an X.25 network can carry data, it is possible to concurrently establish the initial X.25 network connection and the host-host protocol connection. Because of this no additional packets beyond those required by the host-host protocol are needed to initially establish the network connection. There is, however, some packet overhead associated with connection oriented networks, since additional packets are necessary to reestablish broken network connections within the host-host connection.

3.2 Header Overhead

In order to compare percentage overhead due to packet headers when using the two different subnets, we have to make certain assumptions regarding segment and packet lengths. We will assume that packets consist of 128 octets of data and/or segment header, while segments including their headers are 512 octets long. This is consistent with both standard X.25 packet lengths, and the minimum TCP segment length which hosts must be able to handle. Under these assumptions, each segment is sent in four packets and each packet is filled. We also consider the limiting case of very long segments, where only packet headers have a noticable effect.

We first examine the header lengths for segments and datagrams where the underlying network is datagram oriented. We have assumed that a host-host protocol connection number is identical to the concatenation of the complete source and destination process addresses. We do not, however, want the full 12 octet field to appear in every segment header in addition to the 8 octets of address in each datagram header. We thus split this field, as is done in TCP, putting the host addresses in the datagram header for network addressing, and the local process numbers in the segment header for later reassembly of the connection number.

The datagram header must include the following fields with the standard lengths we indicated in Section 2.4.

1. Source and Destination Host Addresses: 8 octets

2. Type of Service: 1 octet

3. Identification Number: 4 octets

4. Control Bits and Format Indicator: 1 octet

5. Time to Live: 1 octet

The host-host protocol header must include the following fields which it needs in addition to the host addresses.

1. Local Process Addresses: 4 octets

2. Sequence Number, Acknowledgement Number and Window Size: 5 octets

3. Segment Length: 2 octets

4. Checksum: 2 octets

5. Control Bits and Format Designator: 1 octet

Thus the segment consisting of 512 octets requires a total of 512 + 4 * 15 = 572 octets to transfer 512 - 14 = 498 octets of user data, a percentage overhead of 74/572 * 100 = 12.9%. In the limit where segments are very large and only packet header contributes significantly to total header overhead, the percentage is 15/128 =11.7%.

By contrast, for the same segment sent via four X.25 packets, each of which have a 3 octet header, it takes 512 + 4 * 3 = 524 octets to transfer 512 - 14 = 498 octets of user data, for a percentage overhead of 26/524 * 100 = 5.0%. Note in this case that the host addresses necessary to reconstruct the host-host protocol connection number are stored at the host with other information about that connection, and are accessed using the network logical channel number. The limiting percentage overhead for very large segments in this case is 3/128 * 100 = 2.3%.

We can see that the use of an X.25 network below our host-host protocol cuts header overhead on data packets from 11.7% - 12.9% to 2.3% - 5.0%, an appreciable amount.

3.3 Buffer Overhead

If one assumes an X.25 interface for a network is available, then the network must either be using fixed routing, or the network is providing sufficient buffering capacity to resequence correctly transmitted packets. In the latter case, the network need not provide sufficient capacity to handle resequencing of retransmitted packets, but must provide enough buffers to handle variations in transit delay through the network for the initial transmission.

For the host-host protocol, the number and occupancy of required buffers are insensitive to the choice of subnet, as long as the network does not use packet retransmission. This is because the number of buffers depends primarily upon the window size in the end-to-end flow-control mechanism, while the buffer occupancy depends upon the delay time for end-to-end acknowledgements, and the even greater delay involved in waiting for retransmitted packets to arrive.

4. TCP AS A HOST-HOST PROTOCOL

4.1 Differences from the Protocol Model

One of the fundamental differences in design between TCP and the host-host protocol model which we outlined in Section 2 concerns the definition and use of the term segment. We defined a segment to be both the unit of information which carries distinct host-host protocol information in a separate header, and the fixed maximum size unit of retransmission. TCP uses the term segment to refer to the header-carrying unit of information, but it acknowledges user data and retransmits at the octet level. Since acknowledgements can only be returned after a full segment has been received and the end-to-end checks completed, an error still requires a whole segment to be retransmitted. The octet level numbering does, however, allow the destination to indicate that it wants the sender to retransmit a segment containing just part of the data in the rejected segment.

This has two effects upon performance. It means that buffer allocation and segment size may be controlled down to the level of individual octets. This feature may be useful or not, and easy or difficult to implement for a given operating system. Certainly many application process / operating system combinations are better suited to fixed size segments and buffers. There is, moreover, an additional processing overhead to handle this flexibility, and the larger required field lengths for sequence number, acknowledgement number and window size do marginally decrease throughput.

Another difference between TCP and our model is one of nomenclature. We spoke of segments and either network datagrams or packets. TCP logically wraps each segment in a datagram, and then if necessary for the network implementation, breaks the segment into fragments and repeats the datagram header for each fragment. Each fragment carries the datagram identification number plus the fragment offset necessary for rebuilding the datagram (and segment) at the destination host.

Three other differences concern individual fields in the TCP header. The length field is part of the datagram header, rather than the TCP header, resulting in another small increase in the total number of header bits required when fragmentation is necessary. A header checksum is also incorporated within the datagram, which exists in addition to both the frame level CRCs and the segment checksum within the segment header. Finally, there is a field for a pointer to urgent data. This is necessary for the particular interrupt mechanism used by TCP.

4.2 Header Overhead

One effect of all the differences listed above is to increase the total overhead of header bits from 5.0% (for virtual circuit nets) or 12.9% (for datagram subnets) to 16.8% for our same example, since using TCP over the Internet datagram with four fragments requires a total of 512 + 4 * 20 = 596 octets to transfer 512 - 20 = 492 octets of user data. The limiting percentage overhead when segments are very large is 20/128 * 100 = 15.6%, as compared to 2.3% or 11.7% This 15.6% - 16.8% overhead significantly reduces the maximum obtainable throughput within a virtual circuit.

4.3 TCP over X.25

There are two approaches to implementing TCP on top of X.25. One inefficient approach is to concatenate the segment and datagram headers onto the user data segment, and then send this through the net within X.25 packets. In the limit of very large segments this reduces header overhead back down to 2.3%. For our same example, with a reasonably small segment, it takes 512 + 4 * 3 = 524 octets to send 512 - 40 = 472 octets of user data, for a percentge overhead of 9.9%.

The problem with this approach is that almost all of the datagram header information is unnecessary when the datagram is being sent along an X.25 virtual circuit.

A second approach therefore would be to eliminate the datagram header and simply replace it with the X.25 header. The problem with this approach is that the length of the segment is not available in the TCP header, and this piece of data is usually required for standard end-to-end checks of data correctness. The TCP connection number in this case must be constructed from the local process addresses in the header, and the host addresses which are stored at the host and accessed via the X.25 connection number.

In any case, the end-to-end checks of the TCP protocol are not cleanly separable from the IP network transmission mechanism, as is the case with the model presented in Section 2.

5.0 CONCLUSIONS

In this paper we have accepted, as given, that a connection oriented host-host protocol should provide easy, reliable transfer of data from one host to another. We have specified some of the basic features that such a protocol should incorporate and then investigated how the choice of subnet affected the performance of the protocol.

Our study points out the following facts with regard to the conflict between X.25 service and overhead, from the point of view of a host-host protocol:

1. Although a connection oriented network is generally somewhat more expensive to build, from the point of view of a higher level host-host protocol the overhead of network connection maintenance is minimal. Extra packet transmission and delay occur only when a network facility goes down and a network virtual connection has to be rerouted. Under normal conditions, the number of packets transmittted across the network is about the same regardless of the subnet type.

2. The number and occupancy of host buffers is primarily a function of end-to-end flow-control and the number of retransmitted packets, rather than a

characteristic of the underlying network (assuming the network does not do packet retransmission).

3. The reduction in throughput due to segment and packet headers is significantly greater for a datagram network than for an X.25-like network with fixed routing, as shown in Table 1.

This information supports the claim that if a connection oriented subnetwork and a datagram subnet are both available then a host-host protocol will perform better over the X.25-like network.

Another important point in favor of X.25 subnetworks is their universality. The X.25 virtual circuit interface has already become a very widely available network interface, and any host-host protocol which wants to be widely accepted should logically build upon this standard network interface. Note in this context, that while the X.25 standard does provide for datagram service in addition to virtual calls, this is an optional feature and no public data networks currently offer this service.

Our brief discussion of TCP has pointed out its throughput problems over an underlying datagram subnet, and its uneasy fit over existing X.25 networks. TCP and IP were designed to use arbitrarily unreliable networks for data transmission. The resulting design ignores the reality that TCP as a Department of Defence standard protocol will be used above either public data networks, or private networks over which DoD has design control. In both those cases an X.25 interface is either already available or could be required.

Acknowledgement: The authors would like to thank Eric Scace and John Heafner for their many helpful discussions.

REFERENCES

1. International Organization for Standardization, ISO/TC97/SC16 N227, "Reference Model of Open Systems Interconnection", August 1979.

2. DOD Standard Transmission Control Protocol, Information Sciences Institute, University of Southern California, RFC: 761, IEN: 129, January 1980.

3. DOD Standard Internet Protocol, Information Sciences Institute, University of Southern California, RFC: 760, IEN: 128, January 1980.

4. CCITT Recommendation X.25 as approved at CCITT Com.VII Plenary 1980 Feb.

5. Sunshine,C., Formal Techniques for Protocol Specification and Verification, Computer 12, No.9, Sept. 1979, pp20:28.

TABLE I

PERCENTAGE HEADER OVERHEAD

	4 PACKETS/ SEGMENT	LARGE SEGMENTS
HOST-HOST OVER DATAGRAM	12.9%	11.7%
HOST-HOST OVER X.25	5.0%	2.3%
TCP OVER IP	16.8%	15.6%
TCP + IP OVER X.25	9.9%	2.3%
TCP OVER X.25	(6.1%)	(2.3%)

APPENDIX D

BIBLIOGRAPHY

"A Data Transfer Protocol for Remote Database Access"
P. S-C. Wang and S. R. Kimbleton
Proc. Trends and Applications: 1980 --- Computer Network Protocols
May 1980

"A Perspective on Network Operating Systems"
S. R. Kimbleton and R. L. Mandell
Proc. National Computer Conference
1976

"Access Control Mechanisms for a Network Data Manager"
H. M. Wood and S. R. Kimbleton
Proc. National Computer Conference
1979

"Applications and Protocols"
S. R. Kimbleton and P. S-C. Wang
An Advanced Course on Distributed Systems --- Architecture and Implementation
M. Paul and H. Siegert, eds.
Springer-Verlag, New York
1980

"Common Command Language for File Manipulation and Network Job Execution: An Example"
M. L. Fitzgerald
NBS SP 500-37
Aug 1978

"Database Semantic Integrity for a Network Data Manager"
E. Fong and S. R. Kimbleton
Proc. National Computer Conference
1980

"Network Operating Systems --- An Implementation Approach"
S. R. Kimbleton, H. M. Wood, and M. L. Fitzgerald
Proc. National Computer Conference
1978

"The Design and Implementation of the National Bureau of Standards' Network Access Machine (NAM)"
R. Rosenthal and B. D. Lucas
NBS SP 500-35
1978

"XNDM: An Experimental Network Data Manager"
S. R. Kimbleton, P. S-C. Wang, and E. N. Fong
Proc. 4th Berkeley Conference on Distributed Data Management and Computer Networks
Aug 1979